Data Independent Induction over structured networks *

S. J. Creese and A. W. Roscoe

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

Abstract *We extend the classes of network which Data Independent Induction can be used to reason about. Through the use of constants and predicates in the data independent type we build proofs of structured networks' behaviours, where a network's topology need not be as regular as one might expect data independence to imply. These properties hold true independent of the size of the type, and so for arbitrary network size. 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

The value of formal methods in the Industrial Validation Flow is widely recognised, however, it is still not standard practise. Systems are becoming increasingly complex, with time to market being drastically reduced. In order to successfully meet these needs the formal techniques utilised must bring results for large and complicated systems quickly. Whilst the speed can be addressed by training and experience, coupled with well communicated research, the challenge for model checkers is to verify systems of realistic size.

It has been demonstrated by many authors (for example [1, 9, 17]) that induction is a method which can be successfully used in the analysis of large systems; results are shown to be true of systems with unboundedly large numbers of identical nodes. The general form of these methods is for an invariant to be defined, and a model checker used to discharge proof obligations which correspond to the base case and inductive step. In [2, 3, 15] a similar type of structural induction scheme was perfected for CSP, utilising the FDR tool to discharge the various proof obligations. The main limitation on this type of structural induction for processes is that it can only address systems in which the individual components are completely independent of the overall structure of the system: adding an extra process does not change the semantics of the components that were already there. Thus, it would not be possible to use induction to reason about a network of nodes where each node knows the identities of the other nodes in the network. In that case an *n+1* node network cannot be constructed by adding one more node to the n node version, since nodes in the n node version would have to be adjusted in order to give them the knowledge of the additional nodes' existence.

It is this limitation on induction which is addressed by the *Data Independent Induction* technique. Specifically, data independent reasoning is used in combination with induction to achieve results for systems of components which are identical in every way except that they require knowledge of each other's existence[1]. See [4, 5, 6] for examples of its use.

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

[1] In general the method applies to situations where

The technique is implemented using the process algebra CSP[8, 14] to model systems, and the FDR model checker tool [7] to help reason about them, though we have no doubt that our ideas would work in other contexts as well. 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. Due to space limitations, this paper concentrates on the technique and ideas involved, keeping actual CSP script to a minimum.

2 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. It must not be possible for the program to extract information from T in other ways that could be used to influence its control flow. This forbids the use of constructs such as `card(S)` (the size of the set `S`). Many communications protocols are data independent since they simply pass around data ensuring that only the intended recipients are able to extract it; their behaviour is entirely independent of the data itself. To date research in the field has centred around the consideration of these particular types, i.e. data which is passed around and stored in concurrent systems.

Data independence ideas have been developed for a number of notations, having first been studied formally in [16], but we use here the theory developed for CSP by Lazic and Roscoe (see [10, 11] and Section 15.2 of [14]). They have developed theorems which state that for certain (data independent) systems it is possible to verify that the system possesses properties, for all instances of its independent types, by performing a specific finite number of refinement 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.

In many cases processes satisfy structural conditions which result in the threshold B on the size of type T, being as small as 1 or 2. However, the types of processes we will be considering require us to perform calculations based on the the variable content of the program to generate thresholds.

The various theorems, definitions and threshold calculations of data independence can be found in [10, 11, 14]. A formulation of some of this work in a very general and largely language-independent setting can be found in [13]. However, of particular relevance to the techniques presented in this report 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 a verification should establish a property for all possible interpretations. The predicate must be a function into a fixed finite type, which in our case will usually be $\{\mathit{true}, \mathit{false}\}$ (otherwise the problem would become intractably infinite).

The definition of data independence forbids the use of parallel compositions indexed by T,

the type indexes a group of processes, rather than being just the node names. In [6], it was used to prove properties of a particular multicasting protocol node with arbitrary downstream branching. This allowed a proof for arbitrary tree networks, where structural induction [2, 3] had only dealt with trees of nodes with fixed degrees of branching. In this case the type indexed the nodes immediately downstream from a fixed one.

since in general this has much the same effect as allowing T to be counted directly. Therefore the core theorems of data independence only allow us to prove things about networks whose topology (at least beyond some finite limit) does not change with the size of T. The method of data independent induction combines two techniques, neither of which can handle this type of network by itself, into a single method that can.

3 Data Independent Induction

In verifying a realistic distributed system we typically want to prove (1) for a large size of T: that a system which is built as the indexed parallel composition over a type T of node processes `N(x)`, where `x` is the identity of the particular node, satisfies a specification `Spec`:

$$\texttt{Spec} \;\sqsubseteq\; \texttt{|| x:T @ [A(x)] N(x)} \quad (1)$$

The nodes are synchronising on the common elements in their respective alphabets, given by `A(x)`. All nodes in the system are identical apart from their unique identities. Both the specification and the nodes treat elements of the type T data independently.

Data Independent Induction provides us with a way of proving properties of networks like that in (1). The proof is broken into two phases: for all $S \subseteq T$

$$\begin{array}{l}\texttt{Superstate(S)} \\ \quad \sqsubseteq \;\; \texttt{(|| x:S @ [A(x)] N(x))}\end{array} \quad (2)$$

$$\texttt{Spec} \;\sqsubseteq\; \texttt{Superstate(T)} \quad (3)$$

Our induction hypothesis is that a (usually sequential) process represents partially constructed versions of the system under consideration, (2) above. It should be data independent in T, should be refined by the partially constructed network it models, and when complete should refine `Spec`. As this process is often a representation of the state of a parallel system we term it `Superstate(S)`, where `S` is the (nonempty) subset of T that constitutes the partial network (`S` is treated as a predicate on T). If an appropriate `Superstate` has been constructed, then (3) presents no new theoretical challenges as it already fits within the established theory of data independence: we can reasonably expect it to be verifiable independently of T by finding the appropriate threshold and doing the necessary refinement checks. These two statements, if true, are of course sufficient to establish our overall aim (1).

The role of induction comes in establishing (2), which does not come within standard data independence theory as it contains an indexed parallel composition over the type. What we do is to perform an induction on the size of `S` (from 1 up to $|T|$). The base case of the induction is to show that one node refines `Superstate`:

$$\texttt{Superstate(\{c\})} \;\sqsubseteq\; \texttt{N(c)} \quad (4)$$

where the constant symbol `c` represents an arbitrary member of T. The inductive step is to show that for any nonempty `S` and constant `c` outside `S`,

$$\begin{array}{l}\texttt{Superstate(S')} \\ \quad \sqsubseteq \;\; \texttt{Superstate(S)[AA(S)||A(c)]N(c)}\end{array} \quad (5)$$

where `S'` is the union of `S` and `{c}` and `AA(S)` is the alphabet of `Superstate(S)`, which is the union of `{A(x) | x <- S}` (the alphabets of each node).

If we can establish these two things the inductive proof of (2) follows easily: if it is true for `S` then

$$\begin{array}{l}\texttt{Superstate(S')} \\ \quad \sqsubseteq \;\; \texttt{Superstate(S)[AA(S)||A(c)]N(c)} \\ \quad \sqsubseteq \;\; \texttt{(||x:S@[A(x)] N(x))[AA(S)||A(c)]N(c)} \\ \quad \sqsubseteq \;\; \texttt{(||x:S'@[A(x)] N(x))}.\end{array}$$

The second line is from the monotonicity of the parallel operator and induction, and the final one from the definition of indexed parallel.

We can thus conclude that the truth of (4) and (5) (for arbitrary T, `S` and `c`) imply that (2) holds for all nonempty S and finite T. Note that this conclusion actually requires not one induction, but rather demonstrates that an infinite number of different inductions (one for each T) all work.

Fortunately, both (4) and (5) are statements that can be proved using data independence extended by predicate symbols and constants, as we can naturally treat `S` (the subset of the network present in a partially-built network) as a predicate symbol (mapping a member of T to *true* if it is in the set). Therefore we can reasonably hope to justify these statements for all values of their parameters, and thus be able to conclude (1) from a finite collection of checks which can be verified automatically on FDR. Note that since `Superstate` has to treat elements of the type T data independently, the connection scheme of the network must also be data independent of the type.

4 A ring topology

All the examples of Data Independent Induction presented to date follow the principle that, with the exception of the indexed parallel composition that we induct through, the entire system is data independent. If we stick to this restriction then the range of network topologies that can arise is extremely limited, for the simple reason that the selection of which of the indexed class any process talks to is itself then data independent.

Assume for simplicity that the network has the form `|| x:T @ [A(x)] P(x)` (i.e. one node for each member of T, and no others). Each node `P(x)` has to treat any two others `P(y)` and `P(z)`, such that `y` and `z` are not differentiated by equality with any constant symbol or `x`, or by any predicate symbol, exactly alike. Thus, if it can talk to one of them it can talk to both. The only network that can be created with no constant symbols or predicates is the fully connected one. With a single constant we can have topologies in which a single node is special, and with a predicate symbol we can have networks in which the nodes are divided up into several distinguished classes. Removing the restriction does not change things greatly, and in particular there is no way in which common network structures like chains, rings and grids can be created, since in these each node `P(x)` is connected to a few others whose identity depends functionally upon `x`.

For example, in a ring, `P(x)` is connected to `P(pred(x))` and `P(succ(x))`, respectively its predecessor and successor. The problem is that operations over T like `pred(x)` and `succ(x)` are illegal in data independent programs. Fortunately, just as the technique of data independent induction was evolved to get round the restriction that we were not allowed indexed parallel compositions over T, we can adapt it so as to avoid this new problem some of the time.

We will concentrate on the case of building a ring:

```
|| x:T @ [A(x)] N(x),
```

where both `A(x)` and `N(x)` make appropriate use of the functions `succ` and `pred` (which are each others' inverses and which define a cycle involving the whole of T). For example, where there is just a single channel `c` connecting successive processes in the ring, and each process `N(x)` has channels `in.x` and `out.x` with the outside world we will typically have:

```
A(x) = {|c.x, c.succ(x), in.x, out.x|}
```

It is natural to formulate an induction for a ring that considers successively longer chains, leaving it to a final step to join a chain consisting of all but one of the processes into a ring. So suppose the subset S of T represents a sequence of consecutive processes in the ring: $x_0, \ldots, x_r$ where $x_{i+1} = succ(x_i)$.

Once again we build a `Superstate` process to play the role of inductive hypothesis: it provides a model of the composition of the processes from S. We need to be able to reformulate analogous steps to (4) and (5) from the last section without the use of the illegal operations `succ` and `pred`: this can often be done using a small number of extra constant symbols. Suppose[2]:

- The node process `N(x)` and its alphabet `A(x)` only use these operations applied a single time to the name `x`. This means that a node's behaviour can special-case its own name and those of its immediate

[2] The conditions given here could be modified to take account of other very finitary dependencies between the `N(x)`, `A(x)` and `Superstate`, but the ones stated here are the most obvious possibility.

neighbours, but not others, and the relative positions of other nodes on the ring is not relevant to its behaviour.

- The `Superstate` process can only distinguish which nodes it contains, which are the first and last in the sequence, and what the external neighbours of these are. It cannot know anything else about the topology of the network, or uses of `succ` and `pred` that occur in it or its alphabet definition.

If these are true then the uses of `succ` and `pred` that may occur in the base and step cases of our induction can be replaced by constant symbols: a node process will have constants representing its own name and those of its predecessor and successor: `Node(id,p,s)`, and `Superstate` will take the form:

```
Superstate(S,f,l,bf,al).
```

where `f` and `l` are the first and last nodes in S, and `bf`, `al` are respectively the nodes before the first (i.e. $\texttt{pred}(\texttt{f}) = \texttt{bf}$) and after the last ($\texttt{succ}(\texttt{l}) = \texttt{al}$).

The base case of the induction is simple: a one node network.

```
Superstate({f},f,f,bf,al)
  ⊑  Node(f,bf,al)
```

Since there is only one node in S the constants representing the first and last nodes in this partially constructed chain must be identical.

The step case is then that, for all nonempty contiguous proper subsets S of T of which `f`, `l` are the first and last members, $\texttt{bf} = \texttt{pred}(\texttt{f})$, $\texttt{al} = \texttt{succ}(\texttt{l})$ and $\texttt{aal} = \texttt{succ}(\texttt{al})$,

```
Superstate(S',f,al,bf,aal)
  ⊑  Superstate(S,f,l,bf,al)
     [AA(S)||A(al)]
     Node(al,l,aal)
```

where $S' = S \cup \{\texttt{al}\}$. In other words we are extending S by adding in the next process round the ring (`al`) and appropriately re-jigging the constants.

This covers all the cases up to and including the complete ring, though the roles of the constants in `Superstate(S,f,l,bf,al)` become somewhat anomalous in the case of a complete ring. It may be better to have a separate "finalisation" case, in which we we replace the `Superstate(S',f,al,bf,aal)` process on the left-hand side, when $S' = T$, by the data independent specification that we want to prove of the complete ring.

There are a number of special cases where some of the constants are equal to each other. The first of these relates to short rings – length 4 or less – which simply become specific simple cases to check (with no need for thresholds). The others are the times near the start and end of the induction: we need to consider separately the steps from 1 to 2 (when $\texttt{f} = \texttt{l}$) and from $n-2$ to $n-1$ (when $\texttt{aal} = \texttt{bf}$). These two cases, the main step case and the base and finalisation cases give five different cases to consider using data independence, all of which may have different thresholds. The important point, however, is that we can once again hope to verify a property of all sizes of network using a finite number of finite refinement checks.

It is straightforward to extend Data Independent Induction to encompass networks in which internal communications are hidden and `Superstate` models partial networks in which the appropriate fraction of the hiding has occurred. The inductive step now usually hides communications between `Superstate` and the extra node.

We only have space for a small example: consider a simple ring network in which each process has the right to broadcast one message at a time to all the other nodes. `N(x)` is a typical node:

```
N(x) = in.x?m -> c.succ(x).x!m -> E(x)
       [] c.x?y?m ->
            if x==y
            then N(x)
            else out.x!y.m ->
                    c.succ(x)!y.m -> N(x)

E(x) = c.x?y?m ->
            if x==y
            then N(x)
            else out.x!y.m ->
                    c.succ(x)!y.m -> E(x)
```

Messages are passed in one direction round the ring, and each one that arrives is output locally and passed on unless it was originated here, in which case it is removed. A node can only accept a new message to broadcast when it has no current expected broadcast.

If these processes are put together in parallel they can deadlock: if all users simultaneously insert a message no further progress can be made. Deadlock can be avoided by extra buffering. One way to do this, which would hardly change the following analysis at all, would be to put two processes at each node N(x), in which the c.succ(x) has been replaced by d.x, in parallel with a one-place buffer from d.x to c.succ(x).

In either case we would evidently expect the broadcasts sent by each process to arrive in turn at each individual destination. A complete description of the ring's (and partial rings') behaviour would not be something we can verify by our method because the specification would not be data independent: the broadcasts emerge from the nodes in ring order, and that is not something we can capture. However the behaviour that links a pair of nodes is accessible. (Of course, having proved the property for an arbitrary pair of nodes, we will really have proved it for all such pairs.) Suppose p and q are two nodes with p $\neq$ q. Since clearly we can build a ring of identical processes starting from any point, we can make life a bit easier for ourselves by starting it at p. The Superstate process we define comes in two parts: one that describes behaviour relevant to communications between p and q, and one that deals with all the rest.

The first part has alphabet consisting of those events directly relevant to communications from p to q, namely {|in.p,out.q.p|} together with the related ring communications: for the segment of the ring whose first element is p and whose last is pred(al), these are {|c.q.p,c.al.p|}. If the union of these two sets is ASS, the first part of the Superstate process is defined as SS(S,p,q,al). In the following definitions, the three uses of CHAOS(ASS) all deal with cases that can arise in partially completed rings, but disappear once the whole system is put together.

```
SS(S,p,q,al) = c.p.p?m -> CHAOS(ASS)
               [] in.p?z ->
                   if S(q)
                   then SS'(S,p,q,al,z)
                   else SS''(S,p,q,al,z)

SS'(S,p,q,al,y) = c.p.p?z -> CHAOS(ASS)
                  [] out.q.p.y ->
                        SS''(S,p,q,al,y)

SS''(S,p,q,al,y) = c.p.p?z -> CHAOS(ASS)
                   [] c.al.p.y ->
                         S'''(S,p,q,al)

SS'''(S,p,q,al) = c.p.p?z -> SS(S,p,q,al)
```

The other part of Superstate is just CHAOS(A), where A is the complete alphabet of the partial ring. The two are put in parallel, synchronising on ASS. The corresponding specification for the completed network is just the process

```
Spec = in?p?z -> out.q!p.z -> Spec
```

having alphabet {| in.p,out.q.p|}, in similar parallel composition with CHAOS({| in, out |}). The scheme of induction we set out above, hiding the channels c.x as they become internalised within S, then goes through precisely. There are in fact two data independent types here: the important one is the one of node names, but the type of message contents is also one and falls easily within the scope of the results which allow a threshold of 2.

5 More complex networks

We can similarly deal with properties of more complex networks than rings and chains. However if, in the partial networks we build, the places we can connect to other nodes are not governed by a constant bound (as they are in a ring) then these need to be described via one or more predicate symbols. The essential feature we require, however much the connection structure of a network fails to be data independent, is that this can be factored out when we perform an individual inductive step.

We are confident that it is possible, by this means, to prove properties of networks with just about any arbitrary topology. The class of problems we can handle is limited to ones where there is a reasonable and finite-state `Superstate` process. The investigation of the theoretical and practical limits of our method remains a topic for further work.

Acknowledgements

We would like thank Ranko Lazić for his work on data independence.

References

[1] E.M. Clarke and O. Grumberg and S. Jha, *Verifying Parameterized Networks using Abstraction and Regular Languages*, Proceedings of CONCUR'95, 395-407, LNCS 962, 1995.

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

[3] S.J. Creese and J. Reed, *Verifying End-to-End Protocols Using Induction with CSP/FDR*, Proceedings of the Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'99), April 1999, San Juan, Puerto Rico. LNCS Volume 1586.

[4] S.J. Creese and A.W. Roscoe, *TTP: A case study in combining induction and data independence*, Oxford University Computing Laboratory Technical Report, PRG-TR-1-99.

[5] S.J. Creese and A.W. Roscoe, *Verifying an infinite family of inductions simultaneously using data independence and FDR*, In Formal Methods for Protocol Engineering and Distributed Systems, the proceedings of Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV'99), October 5-8 1999, Beijing, China. Published by Kluwer Academic Publishers.

[6] S.J. Creese and A.W. Roscoe, *Formal Verification of Arbitrary Network Topologies*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), June 28 - July 1 1999, Las Vegas, Nevade, USA. Volume II. Published by CSREA Press.

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

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

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

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

[11] 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.

[12] R.S. Lazić and A.W. Roscoe, *Data independence with predicate symbols*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), June 28 - July 1 1999, Las Vegas, Nevade, USA. Volume I. Published by CSREA Press.

[13] Ranko Lazić and David Nowak. *A Unifying Approach to Data-independence*, In Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), Lecture Notes in Computer Science. Springer-Verlag, August 2000.

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

[15] 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.

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

[17] 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.