Hierarchical compression for model-checking CSP *or* How to check 10^{20} dining philosophers for deadlock

A.W. Roscoe[*,†] P.H.B. Gardiner[†] M.H. Goldsmith[†] J.R. Hulance[†] D.M. Jackson[†] J.B. Scattergood[*]

1 Introduction

FDR (Failures-Divergence Refinement) [6] is a model-checking tool for CSP [10]. Except for the recent addition of determinism checking [20, 22] (primarily for checking security properties) its method of verifying specifications is to test for the refinement of a process representing the specification by the target process. The presently released version (FDR 1) uses only *explicit* model-checking techniques: it fully expands the state-space of its processes and visits each state in turn. Though it is very efficient in doing this and can deal with processes with approximately 10^7 states in about 4 hours on a typical workstation, the exponential growth of state-space with the number of parallel processes in a network represents a significant limit on its utility. A new version of the tool (FDR 2) is at an advanced stage of development at the time of writing (February 1995) which will offer various enhancements over FDR 1. In particular, it has the ability to build up a system gradually, at each stage compressing the subsystems to find an equivalent process with (hopefully) many less states. By doing this it can check systems which are sometimes exponentially larger than FDR 1 can – such as a network of 10^{20} (or even 10^{1000}) dining philosophers.

This is one of the ways (and the only one which is expected to be released in the immediate future) in which we anticipate adding direct *implicit* model-checking capabilities to FDR. By these means we can certainly rival the sizes of systems analysed by BDD's (see [2], for example) though, like the latter, our implicit methods will certainly be sensitive to what example they are applied to and how skillfully they are used. Hopefully the examples later in this paper will illustrate this.

The idea of compressing systems as they are constructed is not new, and indeed it has been used in a much more restricted sense in FDR for several

*Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford
†Formal Systems (Europe) Ltd,3 Alfred Street, Oxford

years (applying bisimulation at the boundary between its low and high-level processes). The novelty of this paper consists in several of the specific compressions described and in their use in the context of FDR which differs from most other model-checking tools in (i) being based on CSP and (ii) being a refinement checker which compares two CSP processes rather than having the specification written in a different language such as μ-calculus or temporal logic.

The ideas presented in this paper are closely related to those of [8] (whose *interface specifications* – restrictions based on contexts – translate very naturally and usefully to the context of CSP), and also of [3] since we will make considerable use of optimisations resulting from restrictions to the sub-alphabet of interest (which in the important case of deadlock turns out to be the empty set). Most of the literature relates to compressions with respect to strong equivalences such as observational equivalence and bisimulation. The most similar work to our own, because it relates to the weaker, CSP style, equivalences is that of Valmari, for example [12, 25].

The main ideas behind FDR were introduced in a paper in the Hoare *Festschrift* [19] as, indeed, was part of the theory behind this compression.

In this paper we will introduce the main compression techniques used by FDR2 and give some early indications of their efficiency and usefulness.

2 Two views of CSP

The theory of CSP has classically been based on mathematical models remote from the language itself. These models have been based on observable behaviours of processes such as traces, failures and divergences, rather than attempting to capture a full operational picture of how the process progresses.

On the other hand CSP can be given an operational semantics in terms of labelled transition systems. This operational semantics can be related to the mathematical models based on behaviour by defining *abstraction* functions that 'observe' what behaviours the transition system can produce. Suppose Φ is the abstraction function to one of these models. An abstract operator op and the corresponding concrete/operational version $\mathbf{op}$ are congruent if, for all operational processes $\mathbf{P}$, we have $\Phi(\mathbf{op}(\mathbf{P})) = op(\Phi(\mathbf{P}))$. The operational and denotational semantics of a language are congruent if all constructs in the language have this property, which implies that the behaviours predicted for any term by the denotational semantics are always the same as those that can be observed of its operational semantics. That the standard semantics of CSP are congruent to a natural operational semantics is shown in, for example, [18].

Given that each of our models represents a process by the set of its possible behaviours, it is natural to represent refinement as the reduction of these options: the reverse containment of the set of behaviours. If P refines Q we write $Q \sqsubseteq P$, sometimes subscripting $\sqsubseteq$ to indicate which model the refinement it is respect to.

In this paper we will consider three different models – which are the three that FDR supports. These are

- The *traces* model: a process is represented by the set of finite sequences of communications it can perform. $traces(P)$ is the set of P's (finite) traces.

- The *stable failures* model: a process is represented by its set of traces as above and also by its stable failures (s, X) pairs where s is a finite trace of the process and X is a set of events it can refuse after s which (operationally) means coming into a state where it can do no internal action and no action from the set X. *failures*(P) is the set of P's stable failures in this sense. (This model is relatively new; it is introduced in [11]. The concepts behind it will, however, be familiar to anyone well-versed in CSP. It differs from those of [12] in that it entirely ignores divergence.)

- The *failures/divergences* model [1]: a process P *diverges* when it performs an infinite unbroken sequence of internal actions. The set $divergences(P)$ is those traces *after* or *during* which the process can diverge (this set is always suffix closed). In this model a process is represented by $divergences(P)$ and a modified set of failures in which after any divergence the set of failures is extended so that we do not care how the process behaves

$$failures_{\perp}(P) = failures(P) \cup \{(s, X) \mid s \in divergences(P)\}$$

  This is done both because one can argue that a divergent process looks from the outside rather like a deadlocked one (i.e., refusing everything) and because the technical problems of modelling what happens past divergence are not worth the effort.

We will also only deal with the case where the overall alphabet of possible actions is finite, since this makes the model a little more straightforward, and is an obvious prerequisite to model-checking.

All three of these models have the obvious congruence theorem with the standard operational semantics of CSP. In fact FDR works chiefly in the operational world: it computes how a process behaves by applying the rules of the operational semantics to expand it into a transition system. The congruence theorem are thus vital in supporting all its work: it can only claim to prove things about the abstractly-defined semantics of a process because we happen to know that this equals the set of behaviours of the operational process FDR works with.

The congruence theorems are also fundamental in supporting the hierarchical compression which is the main topic of this paper. For we know that, if $C[\cdot]$ is any CSP context, then the value in one of our semantic models of $C[P]$ depends only on the value (in the same model) of P, not on the precise way it is implemented. Therefore, if P is represented as a member of a transition system, and we intend to compute the value of $C[P]$ by expanding it as a transition system also, it may greatly be to our advantage to find another representation of P with fewer states. If, for example, we are combining processes P and Q in parallel and each has 1000 states, but can be compressed to 100, the compressed composition can have no more than 10,000 states while the uncompressed one may have up to 1,000,000.

3 Generalised Transition Systems

A labelled transition system is usually deemed to be a set of (effectively) structureless nodes which have visible or τ transitions to other nodes. ¿From the point of view of compression in the stable failures and failures/divergences models, it is useful to enrich nodes by a set of minimal acceptance sets and a divergence labelling. We will therefore assume that there are functions that map the nodes of a *generalised labelled transition system* (GLTS) as follows:

- *minaccs*(P) is a (possibly empty) set of incomparable (under subset) subsets of Σ (the set of all events). $X \in$ *minaccs*(P) if and only if P can stably accept the set X, refusing all other events, and can similarly accept no smaller set. Since one of these nodes is representing more than one 'state' the process can get into, it can have more than one minimal acceptance. It can also have τ actions in addition to minimal acceptances (with the implicit understanding that the τs are not possible when a minimal acceptance is). However if there is no τ action then there must be at least one minimal acceptance, and in any case all minimal acceptances are subsets of the visible transitions the state can perform.

  minaccs(P) represents the stable acceptances P can make *itself.* If it has τ actions then these might bring it into a state where the process can have other acceptances (and the environment has no way of seeing that the τ has happened), but since these are not performed by the node P but by a successor, these minimal acceptances are not included among those of the node P.

- *div*(P) is either true or false. If it is true it means that P can diverge – possibly as the result of an infinite sequence of implicit τ-actions within P. It is as though P has a τ-action to itself. This allows us to represent divergence in transition systems from which all explicit τ's have been removed.

A node P in a GLTS can have multiple actions with the same label, just as in a standard transition system.

A GLTS combines the features of a standard labelled transition system and those of the normal form transition systems used in FDR 1 to represent specification processes [19]. These have the two sorts of labelling discussed above, but are (apart from the nondeterminism coded in the labellings) deterministic in that there are no τ actions and each node has at most one successor under each $a \in \Sigma$.

The structures of a GLTS allow us to compress the behaviour of all the nodes reachable from a single P under τ actions into one node:

- The new node's visible actions are just the visible transitions (with the same result state) possible for any Q such that $P \xrightarrow{\tau} {}^*Q$.
- Its minimal acceptances are the smallest sets of visible actions accepted by any stable Q such that $P \xrightarrow{\tau} {}^*Q$.
- It is labelled divergent if, and only if, there is an infinite τ-path (invariably containing a loop in a finite graph) from P.
- The new node has no τ actions.

It is this that makes them useful for our purposes. Two things should be pointed out immediately

1. While the above transformation is valid for all the standard CSP equivalences, it is not for most stronger equivalences such as refusal testing and observational/bisimulation equivalence. To deal with one of these either a richer structure of node, or less compression, would be needed.
2. It is no good simply carrying out the above transformation on each node in a transition system. It *will* result in a τ-free GLTS, but one which probably has as many (and more complex) nodes than the old one. Just because $P \xrightarrow{\tau} {}^*Q$ and Q's behaviour has been included in the compressed version of P, this does not mean we can avoid including a compressed version of Q as well: there may well be a visible transition that leads directly to Q. One of the main strategies discussed below – diamond elimination – is designed to analyse which of these Q's can, in fact, be avoided.

FDR2 is designed to be highly flexible about what sort of transition systems it can work on. We will assume, however, that it is always working with GLTS ones which essentially generalise them all. The operational semantics of CSP have to be extended to deal with the labellings on nodes: it is straightforward to construct the rules that allow us to infer the labelling on a combination of nodes (under some CSP construct) from the labellings on the individual ones.

Our concept of a GLTS has been discussed before in [19], and is similar to an "acceptance graph" from [4], though the latter is to all intents the same as the normal form graphs used in FDR1 and discussed in [19, 6].

4 Methods of compression

FDR2 uses at least five different methods of taking one GLTS and attempting to compress it into a smaller one.

1. Strong, node-labelled, bisimulation: the standard notion enriched (as discussed in [19] and the same as Π-bisimulations in [4]) by the minimal acceptance and divergence labelling of the nodes. This is computed by iteration starting from the equivalence induced by equal labelling. This was used in FDR1 for the final stage of normalising specifications.

2. τ-loop elimination: since a process may choose automatically to follow a τ-action, it follows that all the processes on a τ-loop (or, more properly, a strongly connected component under τ-reachability) are equivalent.

3. Diamond elimination: this carries out the node-compression discussed in the last section systematically, so as to include as few nodes as possible in the output graph.

4. Normalisation: discussed extensively elsewhere, this can give significant gains, but it suffers from the disadvantage that by going through powerspace nodes it can be expensive and lead to expansion.

5. Factoring by semantic equivalence: the compositional models of CSP we are using all represent much weaker congruences than bisimulation. Therefore if we can afford to compute the semantic equivalence relation over states it will give better compression than bisimulation to factor by this equivalence relation.

There is no need here to describe either bisimulation, normalisation, or the algorithms used to compute them. Efficient ways of computing the strongly connected components of a directed graph (for τ-loop elimination) can be found in many textbooks on algorithm design (e.g., [16]). Therefore we shall concentrate on the other two methods discussed above, and appropriate ways of combining the five.

Before doing this we will show how to factor a GLTS by an equivalence relation on its nodes (something needed both for τ-loop elimination and for factoring by a semantic equivalence). If $\mathcal{T} = (T, \rightarrow, r)$ is a GLTS (r being its root) and $\cong$ is an equivalence relation over it, then the nodes of $\mathcal{T}/\cong$ are the equivalence classes $\overline{n}$ for $n \in T$, with root $\overline{r}$. The actions are as follows:

- If $a \neq \tau$, then $\overline{m} \xrightarrow{a} \overline{n}$ if and only if there are $m' \in \overline{m}$ and $n' \in \overline{n}$ such that $m' \xrightarrow{a} n'$.

- If $\overline{m} \neq \overline{n}$, then $\overline{m} \xrightarrow{\tau} \overline{n}$ if and only if there are $m' \in \overline{m}$ and $n' \in \overline{n}$ such that $m' \xrightarrow{\tau} n'$.

- If $\overline{m} = \overline{n}$, then $\overline{m} \overset{\tau}{\not\longrightarrow} \overline{n}$ but (if we are concerned about divergence) the new node is marked divergent if and only if there is an infinite τ-path amongst the members of $\overline{m}$, or one of the $m' \in \overline{m}$ is already marked divergent.

The minimal acceptance marking of $\overline{m}$ is just the union of those of its members, with non-minimal sets removed.

4.1 Computing semantic equivalence

Two nodes that are identified by strong node-labelled bisimulation are always semantically equivalent in each of our models. The models do, however, represent much weaker equivalences and there may well be advantages in factoring the transition system by the appropriate one. The only disadvantage is that the computation of these weaker equivalences is more expensive: it requires an expensive form of normalisation, so

- there may be systems where it is impractical, or too expensive, to compute semantic equivalence, and
- when computing semantic equivalence, it will probably be to our advantage to reduce the number of states using other compression techniques first – see a later section.

To compute the semantic equivalence relation we require the *entire normal form* of the input GLTS $\mathcal{T}$. This is the normal form that includes a node equivalent to each node of the original system, with a function from the original system which exhibits this equivalence (the map need neither be injective – because it will identify nodes with the same semantic value – nor surjective – because the normal form sometimes contains nodes that are not equivalent to any single node of the original transition system).

Calculating the entire normal form is more time-consuming that ordinary normalisation. The latter begins its normalisation search with a single set (the τ-closure $\tau^*(r)$ of $\mathcal{T}$'s root),but for the entire normal form it has to be seeded with $\{\tau^*(n) \mid n \in T\}$ – usually* as many sets as there are nodes in T. As with ordinary normalisation, there are two phases: the first (pre-normalisation) computing the subsets of T that are reachable under any trace (of visible actions) from any of the seed nodes, with a unique-branching transition structure over it. Because of this unique branching structure, the second phase, which is simply a strong node-labelled bisimulation over it, guarantees to compute a normal form where all the nodes have distinct semantic values. We distinguish between the three semantic models as follows:

- For the traces model, neither minimal acceptance nor divergence labelling is used for the bisimulation.

*If and only if there are no τ-loops.

- For the stable failures model, only minimal acceptance labelling is used.
- For the failures/divergences model, both sorts of labelling are used and in the pre-normalisation phase there is no need to search beyond a divergent node.

The map from T to the normal form is then just the composition of that which takes n to the pre-normal form node $\tau^*(n)$ and the final bisimulation.

The equivalence relation is then simply that induced by the map: two nodes are equivalent if and only if they are mapped to the same node in the normal form. The compressed transition system is that produced by factoring out this equivalence using the rules discussed earlier. To prove that the compressed form is equivalent to the original (in the sense that, in the chosen model, every node m is equivalent to $\overline{m}$ in the new one) one can use the following lemma and induction, based on the fact that each equivalence class of nodes under semantic equivalence is trivially τ-convex as required by the lemma.

LEMMA 1
Suppose $\mathcal{T}$ be any GLTS and let M be any set of nodes in $\mathcal{T}$ with the following two properties

- All members of M are equivalent in one of our three models $\mathcal{C}$.
- M is convex under τ (i.e., if $m, m' \in M$ and m'' are such that $m \xrightarrow{\tau} {}^*m'' \xrightarrow{\tau} {}^*m'$ then $m'' \in M$.

Then let $\mathcal{T}'$ be the GLTS $\mathcal{T}/\equiv$, where $\equiv$ is the equivalence relation which identifies all members of M but no other distinct nodes in $\mathcal{T}$. m is semantically equivalent in the chosen model to $\overline{m}$ (the corresponding node in $\mathcal{T}'$).

PROOF
It is elementary to show that each behaviour (trace or failure or divergence) is one of $\overline{m}$ (this does not depend on the nature of $\equiv$).

Any behaviour of a node $\overline{m}$ of $\mathcal{T}'$ corresponds to a sequence σ of actions

$$\overline{m} = \overline{m_0} \xrightarrow{x_1} \overline{m_1} \xrightarrow{x_2} \overline{m_2} \ldots$$

either going on for ever (with all but finitely many x_i τ's), or terminating and perhaps depending on either a minimal acceptance or divergence marking in the final state. Without loss of generality we can assume that the m_r are chosen so that there is, for each r, m'_{r+1} such that $m_r \xrightarrow{x_r} m'_{r+1}$ and that (if appropriate) the final m_r possesses the divergence or minimal acceptance which the sequence demonstrates. Set $m'_0 = m$, the node which we wish to demonstrate has the same behaviour exemplified by σ.

For any relevant s, define $\sigma \uparrow s$ to be the final part of σ starting at $\overline{m_s}$:

$$\overline{m_s} \xrightarrow{x_{s+1}} \overline{m_{s+1}} \xrightarrow{x_{s+2}} \overline{m_{s+2}} \ldots$$

If M, the only non-trivial equivalence class appears more than once in the final (τ-only) segment of an infinite demonstration of a divergence, then all intermediate classes must be the same (by the τ-convexity of M). But this is impossible since an equivalence class never has a τ action to itself (by the construction of $\mathcal{T}/\equiv$).

Hence, σ can only use this non-trivial class finitely often. If it appears no times then the behaviour we have in $\mathcal{T}'$ is trivially one in $\mathcal{T}$. Otherwise it must appear some last time in σ, as $\overline{m_r}$, say. What we will prove, by induction for s from r down to 0, is that the node m'_s (and hence $m = m'_0$) possesses the same behaviour demonstrated by the sequence $\sigma \upharpoonright s$ in $\mathcal{T}'$.

If the special node M in T' becomes marked by a divergence or minimal acceptance (where relevant to $\mathcal{C}$) through the factoring then it is trivial that some member of the equivalence class has that behaviour and hence (in the relevant models) *all* the members of M do (though perhaps after some τ actions) since they are equivalent in $\mathcal{C}$. It follows that if $\overline{m_r}$ is the final state in σ, then our inductive claim holds.

Suppose $s \leq r$ is not final in σ and that the inductive claim has been established for all i with $s < i \leq r$. Then the node m_s is easily seen to possess in $\mathcal{T}$ the behaviour of $\sigma \upharpoonright s$. If the equivalence class of m_s is not M then $m_s = m'_s$ and there is nothing else to prove. If it is M then since m_s and m'_s are equivalent in $\mathcal{C}$ and m'_s has the behaviour, it follows that m'_s does also. This completes the proof of the lemma.

4.2 Diamond elimination

This procedure assumes that the relation of τ-reachability is a partial order on nodes. If the input transition system is known to be divergence free then this is true, otherwise τ-loop elimination is required first (since this procedure guarantees to achieve the desired state).

Under this assumption, diamond reduction can be described as follows, where the input state-machine is $\mathcal{S}$ (in which nodes can be marked with information such as minimal acceptances), and we are creating a new state-machine $\mathcal{T}$ from all nodes explored in the search:

- Begin a search through the nodes of $\mathcal{S}$ starting from its root N_0. At any time there will be a set of unexplored nodes of $\mathcal{S}$; the search is complete when this is empty.
- To explore node N, collect the following information:
  - The set $\tau^*(N)$ of all nodes reachable from N under a (possibly empty) sequence of τ actions.
  - Where relevant (based on the equivalence being used), divergence and minimal acceptance information for N: it is divergent if any member of $\tau^*(N)$ is either marked as divergent or has a τ to itself.

The minimal acceptances are the union of those of the members of $\tau^*(N)$, with non-minimal sets removed. This information is used to mark N in T.

  - The set $V(N)$ of initial visible actions: the union of the set of all non-τ actions possible for any member of $\tau^*(N)$.
  - For each $a \in V(N)$, the set $N_a = N$ *after* a of all nodes reachable under a from any member of $\tau^*(N)$.
  - For each $a \in V(N)$, the set $min(N_a)$ which is the set of all τ-minimal elements of N_a.

- A transition (labelled a) is added to $\mathcal{T}$ from N to each N' in $min(N_a)$, for all $a \in V(N)$. Any nodes not already explored are added to the search.

This creates a transition system where there are no τ-actions but where there can be ambiguous branching under visible actions, and where nodes might be labelled as divergent. The reason why this compresses is that we do not include in the search nodes where there is another node similarly reachable but demonstrably at least as nondeterministic: for if $M \in \tau^*(N)$ then N is always at least as nondeterministic as M. The hope is that the completed search will tend to include only those nodes that are τ-minimal: not reachable under τ from any other. Notice that the behaviours of the nodes not included from N_a are nevertheless taken account of, since their divergences and minimal acceptances are included when some node of $min(N_a)$ is explored.

It seems counter-intuitive that we should work hard *not* to unwind τ's rather than doing so eagerly. The reason why we cannot simply unwind τ's as far as possible (i.e., collecting the τ-maximal points reachable under a given action) is that there will probably be visible actions possible from the unstable nodes we are trying to bypass. It is impossible to guarantee that these actions can be ignored.

The reason we have called this compression *diamond elimination* is because what it does is to (attempt to) remove nodes based on the diamond-shaped transition arrangement where we have four nodes P, P', Q, Q' and $P \xrightarrow{\tau} P'$, $Q \xrightarrow{\tau} Q'$, $P \xrightarrow{a} Q$ and $P' \xrightarrow{a} Q'$. Starting ¿from P, diamond elimination will seek to remove the nodes P' and Q'. The only way in which this might fail is if some further node in the search forces one or both to be considered.

The lemma that shows why diamond reduction works is the following.

LEMMA 2
Suppose N is any node in $\mathcal{S}$, $s \in \Sigma^*$ and $N_0 \stackrel{s}{\Longrightarrow} N$ (i.e., there is a sequence of nodes $M_0 = N_0, M_1, ..., M_k = N$ and actions $x_1, ..., x_k$ such that $M_i \xrightarrow{x_i} M_{m+1}$ for all i and $s = \langle x_i \mid i = 1, .., n, x_i \neq \tau \rangle$). Then there is a node N' in $\mathcal{T}$ such that $N_0 \stackrel{s}{\Longrightarrow} N'$ in $\mathcal{T}$ and $N \in \tau^*(N')$.

PROOF
This is by induction on the length if s. If s is empty the result is obvious (as

$N_0 \in T$ always), so assume it holds of s' and $s = s'\langle a\rangle$, with $N_0 \stackrel{s}{\Longrightarrow} N$. Then by definition of $\stackrel{s}{\Longrightarrow}$, there exist nodes N_1 and N_2 of S such that $N_0 \stackrel{s'}{\Longrightarrow} N_1$, $N_1 \stackrel{a}{\longrightarrow} N_2$ and $N \in \tau^*(N_2)$.

By induction there thus exists N_1' in T such that $N_0 \stackrel{s'}{\Longrightarrow} N_1'$ in $\mathcal{T}$ and $N_1 \in \tau^*(N_1')$. Since $N_1' \in T$ it has been explored in constructing $\mathcal{T}$. Clearly $a \in V(N_1')$ and $N_2 \in (N_1')_a$. Therefore there exists a member N' of $min((N_1')_a)$ (a subset of the nodes of T) such that $N_2 \in \tau^*(N')$. Then, by construction of $\mathcal{T}$ and since $N \in \tau^*(N_2)$ we have $N_0 \stackrel{s}{\Longrightarrow} N'$ and $N \in \tau^*(N')$ as required, completing the induction.

This lemma shows that every behaviour displayed by a node of $\mathcal{S}$ is (thanks to the way we mark each node of $\mathcal{T}$ with the minimal acceptances and divergence of its τ-closure) displayed by a node of $\mathcal{T}$.

Lemma 2 shows that the following two types of node are certain to be included in $\mathcal{T}$:

- The initial node N_0.
- S_0, the set of all τ-minimal nodes (ones not reachable under τ from any other).

Let us call $S_0 \cup \{N_0\}$ the *core* of S. The obvious criteria for judging whether to try diamond reduction at all, and of how successful it has been once tried, will be based on the core. For since the only nodes we can hope to get rid of are the complement of the core, we might decide not to bother if there are not enough of these as a proportion of the whole. And after carrying out the reduction, we can give a success rating in terms of the percentage of non-core nodes eliminated.

Experimentation over a wide range of example CSP processes has demonstrated that diamond elimination is a highly effective compression technique, with success ratings usually at or close to 100% on most natural systems. To illustrate how diamond elimination works, consider one of the most hackneyed CSP networks: N one-place buffer processes chained together.

$$COPY \gg COPY \gg \ldots COPY \gg COPY$$

Here, $COPY = left?x \longrightarrow right!x \longrightarrow COPY$. If the underlying type has k members then $COPY$ has $k+1$ states and the network has $(k+1)^N$. Since all of the internal communications (the movement of data from one $COPY$ to the next) become τ actions, this is an excellent target for diamond elimination. And in fact we get 100% success: the only nodes retained are those that are not τ-reachable from any other. These are the ones in which all of the data is as far to the left as it can be: there are no empty $COPY$'s to the left of a full one. If $k = 1$ this means there are now $N+1$ nodes rather than 2^N, and if $k = 2$ it gives $2^{N+1} - 1$ rather than 3^N.

4.3 Combining techniques

The objective of compression is to reduce the number of states in the target system as much as possible, with the secondary objectives of keeping the number of transitions and the complexity of any minimal acceptance marking as low as possible.

There are essentially two possibilities for the best compression of a given system: either its normal form or the result of applying some combination of the other techniques. For whatever equivalence-preserving transformation is performed on a transition system, the normal form (from its root node) must be invariant; and all of the other techniques leave any normal form system unchanged. In many cases (such as the chain of $COPY$s above) the two will be the same size (for the diamond elimination immediately finds a system equivalent to the normal form, as does equivalence factoring), but there are certainly cases where each is better.

The relative speeds (and memory use) of the various techniques vary substantially from example to example, but broadly speaking the relative efficiencies are (in decreasing order) τ-loop elimination (except in rare complex cases), bisimulation, diamond elimination, normalisation and equivalence factoring. The last two can, of course, be done together since the entire normal form contains the usual normal form within it. Diamond elimination is an extremely useful strategy to carry out before either sort of normalisation, both because it reduces the size of the system on which the normal form is computed (and the number of seed nodes for the entire normal form) and because it eliminates the need for searching through chains of τ actions which forms a large part of the normalisation process.

One should note that all our compression techniques guarantee to do no worse than leave the number of states unchanged, with the exception of normalisation which in the worst case can expand the number of states exponentially[19, 13]. Cases of expanding normal forms are very rare in practical systems. Only very recently, after nearly four years, have we encountered a class of practically important processes whose normalisation behaviour is pathological. These are the "spy" processes used to seek errors in security protocols [21].

At the time of writing all of the compression techniques discussed have been implemented and many experiments performed using them. Ultimately we expect that FDR2's compression processing will be automated according to a strategy based on a combination of these techniques, with the additional possibility of user intervention.

5 Compression in context

FDR2 will take a complex CSP description and build it up in stages, compressing the resulting process each time. Ultimately we expect these decisions to be at least partly automated, but in early versions the compression directives will be included in the syntax of the target process.

One of the most interesting and challenging things when incorporating these ideas is preserving the debugging functionality of the system. The debugging process becomes hierarchical: at the top level we will find erroneous behaviours of compressed parts of the system; we will then have to debug the pre-compressed forms for the appropriate behaviour, and so on down. On very large systems (such as that discussed in the next section) it will not be practical to complete this process for all parts of the system. Therefore we expect the debugging facility initially to work out subsystem behaviours down as far as the highest level compressed processes, and only to investigate more deeply when directed by the user (through the X Windows debugging facility of FDR).

The way a system is composed together can have an enormous influence on the effectiveness of hierarchical compression. The following principles should generally be followed:

1. Put together processes which communicate with each other together early. For example, in the dining philosophers, you should build up the system out of consecutive fork/philosopher pairs rather than putting the philosophers all together, the forks all together and then putting these two processes together at the highest level.

2. Hide all events at as low a level as is possible. The laws of CSP allow the movement of hiding inside and outside a parallel operator as long as its synchronisations are not interfered with. In general therefore, any event that is to be hidden should be hidden the first time (in building up the process) that it no longer has to be synchronised at a higher level. The reason for this is that the compression techniques all tend to work much more effectively on systems with many τ actions.

3. Hide all events that are irrelevant (in the sense discussed below) to the specification you are trying to prove.

Hiding can introduce divergence, and thereby invalidate many failures/divergences model specifications. However in the traces model it does not alter the sequence of unhidden events, and in the stable failures model does not alter refusals which contain every hidden event. Therefore if only trying to prove a property in one of these models – or if it has already been established by whatever method that one's substantive system is divergence free – the improved compression we get by hiding extra events makes it worthwhile doing so.

We will give two examples of this, one based on the $COPY$ chain example we saw above and one on the dining philosophers. The first is probably typical of the gains we can make with compression and hiding; the second is atypically good.

5.1 Hiding and safety properties

If the underlying datatype T of the $COPY$ processes is large, then chaining N of them together will lead to unmanageably large state-spaces whatever sort of compression is applied to the entire system. For it really does have a lot of distinct states: one for each possible contents the resulting N-place buffer might have. Of course there are analytic techniques that can be applied to this simple example that pin down its behaviour, but we will ignore these and illustrate a general technique that can be used to prove simple safety properties of complex networks. Suppose x is one member of the type T; an obviously desirable (and true) property of the $COPY$ chain is that the number of x's input on channel *left* is always greater than or equal to the number output on *right*, but no greater than the latter plus N. Since the truth or falsity of this property is unaffected by the system's communications in the rest of its alphabet $\{left.y, right.y \mid y \in \Sigma \setminus \{x\}\}$ we can hide this set and build the network up a process at a time from left to right. At the intermediate stages you have to leave the right-hand communications unhidden (because these still have to be synchronised with processes yet to be built in) but nevertheless, in the traces model, the state space of the intermediate stages grows more slowly with n than without the hiding. In fact, with n $COPY$ processes the hidden version compresses to exactly 2^n states whatever the size of T (assuming that this is at least 2).

This is a substantial reduction, but is perhaps not as good as one might ideally hope for. By hiding all inputs other than the chosen one, we are ignoring what the contents of the systems are apart from x, but because we are still going to compose the process with one which will take all of our outputs, these have to remain visible, and the number of states mainly reflects the number of different ways the outputs of objects other than x can be affected by the order of inputting and outputting x. The point is that we do not know (in the method) that the outputs other than x are ultimately going to be irrelevant to the specification, for we are not making any assumptions about the process we will be connected to.

Since the size of system we can compress is always likely to be one or two orders of magnitude smaller than the number of explicit states in the final refinement check, it would actually be advantageous to build this system not in one direction as indicated above, but from both ends and finally compose the two halves together. (The partially-composed system of n right-hand processes also has 2^N states.) Nothing useful would (in this example) be achieved by building up further pieces in the middle, since we only get the simplifying benefit of the hiding from the two ends of the system.

If the (albeit slower) exponential growth of states even after hiding and compressing the actual system is unacceptable, there is one further option: find a network with either less states, or better compression behaviour, that the actual one refines, but which can still be shown to satisfy the specification. In the example above this is easy: simply replace $COPY$ with

$$C_x = (\mu\, p.\mathit{left}.x \longrightarrow \mathit{right}.x \longrightarrow p) \parallel\!\mid CHAOS(\Sigma \setminus \{\mathit{left}.x, \mathit{right}.x\})$$

the process which acts like a reliable one-place buffer for the value x, but can input and output as it chooses one other members of T. It is easy to show that $COPY$ refines this, and a chain of n C_x's compresses to $n+1$ states (even without hiding irrelevant external communications).

In a sense the C_x processes capture the essential reason why the chain of $COPY$'s satisfy the x-counting specification. By being clever we have managed to automate the proof for much larger networks than following the 'dumb' approach, but of course it is not ideal that we have had to be clever in this way.

The methods discussed in this section could be used to prove properties about the reliability of communications between a given pair of nodes in a complex environment, and similar cases where the full complexity of the operation of a system is irrelevant to why a particular property is true.

5.2 Hiding and deadlock

In the stable failures model, a system can deadlock if and only if $P \setminus \Sigma$ can. In other words, we can hide absolutely all events – and move this hiding as far into the process as possible using the principles already discussed.

Consider the case of the N dining philosophers (in a version, for simplicity, without a Butler process). A natural way of building this system up hierarchically is as progressively longer chains of the form

$$PHIL_0 \| FORK_0 \| PHIL_1 \| \ldots \| FORK_{m-1} \| PHIL_m$$

In analysing the whole system for deadlock, we can hide all those events of a subsystem that do not synchronise with any process outside the subsystem. Thus in this case we can hide all events other than the interactions between $PHIL_0$ and $FORK_{N-1}$, and between $PHIL_m$ and $FORK_m$. The failures normal form of the subsystem will have very few states (exactly 4). Thus we can compute the failures normal form of the whole hidden system, adding a small fixed number of philosopher/fork combinations at a time, in time proportional to N, even though an explicit model-checker would find exponentially many states.

We can, in fact, do even better than this. Imagine doing the following:

- First, build a single philosopher/fork combination hiding all events not in its external interface, and compress it. This will (with standard definitions) have 4 states.

- Next, put 10 copies of this process together in parallel, after suitable renaming to make them into consecutive pairs in a chain of philosophers and forks (the result will have approximately 4000 states) and compress it to its 4 states.

- Now rename this process in 10 different ways so that it looks like 10 adjacent groups of philosophers, compute the results and compress it.
- And repeat this process as often as you like...clearly it will take time linear in the number of times you do it.

By this method we can produce a model of 10^N philosophers and forks in a row in time proportional to N. To make them into a ring, all you would have to do would be to add another row of one or more philosophers and forks in parallel, synchronising the two at both ends. Depending on how it was built (such as whether all the philosophers are allowed to act with a single handedness) you would either find deadlock or prove it absent from a system with doubly exponential number of states.

On the prototype version of FDR2, we have been able to use this technique to demonstrate the deadlock of 10^{1000} philosophers in 15 minutes, and then to use the debugging tool described earlier to tell you the state of any individual one of them (though the depth of the parse tree even of the efficiently constructed system makes this tedious). Viewed through the eyes of explicit model-checking, this system has perhaps $7^{10^{1000}}$ states. Clearly this simply demonstrates the pointlessness of pure state-counting.

This example is, of course, extraordinarily well-suited to our methods. What makes it work are firstly the fact that the networks we build up have a constant-sized external interface (which could only happen in networks that were, like this one, chains or nearly so) and have a behaviour that compresses to a bounded size as the network grows.

On the whole we do not have to prove deadlock freedom of quite such absurdly large systems. We expect that our methods will also bring great improvements to the deadlock checking of more usual size ones that are not necessarily as perfectly suited to them as the example above.

6 Related Work

A wide range of automated systems have been proposed for the analysis of state-transition systems [5, 7, 14, 17] and it is instructive to examine where FDR, as an industrial product, falls in the range of possibilities identified by academic research. The more flexible tools, like the Concurrency Workbench of [5], permit a wide range of semantic operations to be carried out in those formalisms which exhibit less consensus about the central semantic models. Choosing an alternative approach, systems constructed to decide specific questions about suitably constructed finite-state representations can achieve much greater performance [9, 23].

In designing FDR we make a compromise between these extremes: the CSP language provides a flexible and powerful basis for describing problems, yet by concentrating on the standard CSP semantics we are able to achieve acceptable performance levels. Milner's scheduling problem (used as a benchmark in [5]) can be reduced to CSP normal form in around 3s for seven clients and around 45s for ten. (The admittedly somewhat outdated figure for bisimulation minimization using the Concurrency Workbench is 2000s for seven clients, and the ten client problem was too large to be considered.) Furthermore, the flexibility of CSP as a specification language removes much of the need for special-case algorithms to detect deadlock or termination (such as those proposed as additions to Winston in [15]).

Perhaps the most comparable approach is that taken by the SMV system [14], which decides whether CTL logical specifications are satisfied by systems expressed as state-variable assignments. The BDD representation used by SMV can encode very large problems efficiently, although as with any implicit scheme its effectiveness can vary with the manner in which a system is described: in this regard we hope that identifying candidate components for compression or abstraction may prove easier in practice than arranging that state variables respect a regular logical form. Unlike SMV, a key feature of FDR is its use of a process algebra for both specification and design, encouraging step-wise refinement and the combination of automatic verification with conventional proof. The underlying semantic model, and the extrinsic nature of FDR2 compression can, of course, be applied to any notation or representation which can be interpreted within the FDR framework. FDR2 is designed to facilitate such extension.

7 Conclusions

We have given details of how FDR2's compression works, and some simple examples of how it can expand the size of problem we can automatically check. At the time of writing we have not had time to carry out many evaluations of this new functionality on realistic-sized examples, but we have no reason to doubt that compression will allow comparable improvements in these.

It is problematic that the successful use of compression apparently takes somewhat more skill than explicit model-checking. Only by studying its use in large-scale case studies can we expect to assess the best ways to deal with this – by automated tactics and transformation, or by design-rule guidance to the user. In any case much work will be required before we can claim to understand fully the capabilities and power of the extended tool.

Acknowledgements

As well as our owing him a tremendous debt for his development of CSP, on which all this work is based, it was a remark by Tony Hoare that led the first author to realise how our methods could check the exponential systems of dining

philosophers described in this paper.

We would like to thank the referees for some helpful remarks, in particular for pointing out the need for Lemma 1.

The work of Roscoe and Scattergood was supported in part by a grant from the US Office of Naval Research.

References

[1] S.D. Brookes and A.W. Roscoe, *An improved failures model for communicating processes,* in Proceedings of the Pittsburgh seminar on concurrency, Springer LNCS 197 (1985), 281-305.

[2] J.R. Burch, E.M. Clarke, D.L. Dill and L.J. Hwang, *Symbolic model checking:* 10^{20} *states and beyond*, Proc. 5th IEEE Annual Symposium on Logic in Computer Science, IEEE Press (1990).

[3] E.M. Clarke, D.E. Long and K.L.MacMillan, *Compositional Model Checker*, Proceedings of LICS 1989.

[4] R. Cleaveland and M.C.B. Hennessy, *Testing Equivalence as a Bisimulation Equivalence*, FAC **5** (1993) pp1–20.

[5] R. Cleaveland, J. Parrow and B. Steffen, *The Concurrency Workbench: A semantics-based verification tool for finite state systems*, ACM TOPLAS Vol.15, N.1, 1993, pp.36-72.

[6] Formal Systems (Europe) Ltd., *Failures Divergence Refinement* User Manual and Tutorial, version 1.4 1994.

[7] J.-C. Fernandez *An implementation of an efficient algorithm for bisimulation equivalence.*, Science of Computer Programming 13: 219–236, 1989/1990.

[8] S.Graf and B. Steffen, *Compositional Minimisation of Finite-State Systems*, Proceedings of CAV1990 (LNCS 531).

[9] J.F. Groote and F. Vaandrager, *An efficient algorithm for branching bisimulation and stuttering equivalence*, Proc. 17th ICALP, Springer-Verlag LNCS 443, 1990.

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

[11] L. Jategoankar, A Meyer and A.W. Roscoe, *Separating failures from divergence*, in preparation.

[12] R. Kaivola and A Valmari *The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic* in Proc CONCUR '92 (LNCS 630).

[13] P.C. Kanellakis and S.A. Smolka, *CCS expressions, Finite state processes and three problems of equivalence*, Information and Computation **86**, 43-68 (1990).

[14] K.L. McMillan, Symbolic Model Checking, Kluwer, 1993.

[15] Malhotra, J., Smolka, S.A., Giacalone, A. and Shapiro, R., *Winston: A Tool for Hierarchical Design and Simulation of Concurrent Systems.*, In *Proceedings of the Workshop on Specification and Verification of Concurrent Systems*, University of Stirling, Scotland, 1988.

[16] K. Melhorn *Graph Algorithms and NP Completeness*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag 1984.

[17] J. Richier, C. Rodriguez, J. Sifakis and J. Voiron, *Verification in XESAR of the Sliding Window Protocol*, Proc. of the 7th IFIP Symposium on Protocol Specification, Testing, and Verification, North-Holland, Amsterdam, 1987.

[18] A.W. Roscoe, *Unbounded Nondeterminism in CSP*, in 'Two Papers on CSP', PRG Monograph PRG-67. Also Journal of Logic and Computation **3**, 2 pp131-172 (1993).

[19] A.W. Roscoe, *Model-checking CSP*, in A Classical Mind: Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed.) Prentice-Hall 1994.

[20] A.W. Roscoe, CSP and determinism in security modelling to ap pear in the proceedings of 1995 IEEE Symposium on Security and Privacy.

[21] A.W. Roscoe, *Modelling and verifying key-exchange protocols using CSP and FDR*, to appear in the proceedings of CSFW8 (1995), IEEE Press.

[22] A.W. Roscoe, J.C.P. Woodcock and L. Wulf, *Non-interference t hrough determinism*, Proc. ESORICS 94, Springer LNCS 875, pp 33-53.

[23] V. Roy and R. de Simone, *Auto/Autograph*, In Proc. Computer-Aided Verification '90, American Mathematical Society, Providence, 1991.

[24] J.B. Scattergood, *A basis for CSP tools*, To appear as Oxford University Computing Laboratory technical monograph, 1993.

[25] A. Valmari and M. Tienari *An improved failures equivalence for finite-state systems with a reduction algorithm*, in Protocol Specification, Testing and Verification XI, North-Holland 1991.