Five ways to use induction and symmetry in the verification of networks of processes by model-checking

M. Calder and A. Miller

Department of Computing Science
University of Glasgow
Glasgow, Scotland.

The verification of networks of processes by model-checking is discussed. Five classes of problem in which either symmetry or induction (or both) can be used to solve problems of state-space explosion, case explosion, or generalisation are considered. Examples are given. Recent results in the field are discussed in relation to the proposed classification.

keywords: networks of communicating processes; model checking; induction; symmetry.

1 Introduction

Model checking is a technique which allows reasoning about the temporal behaviour of networks of processes by checking that a finite-state model satisfies a specification. That is, reasoning about networks involves showing

$$M(P_k) \models \phi$$

where $M(P_k)$ represents a finite-state concurrent system P_k and ϕ is a temporal logic formula. The system P_k is the parallel composition $p_1||p_2||\ldots||p_k$ of processes $p_1, p_2, \ldots, p_k$, possibly with "environment" q. The p_i (and q) may or may not communicate, if the former, communication may be either synchronous or asynchronous.

While model-checking is a natural technique for networks of processes, because the semantics are usually given in terms of transition systems, the applicability of model-checking is limited due to:

- *state-space explosion*, the combinatorial explosion in the number of states
- *case explosion*, the combinatorial explosion in the number of cases to be checked,
- *lack of inference*, the inability to infer properties of a system P_{k+1} from properties of of P_k,
- *lack of generalisation*, the inability to generalise properties to members of the infinite family $\{P_n\}_{n=1}^{\infty}$.

In this paper we consider the contribution of two techniques:

- *symmetry*, and
- *induction*

to overcome these limitations and extend the applicability of model-checking.

Our goal is to bring together numerous results in symmetry and induction in one coherent framework and define five classes of problem in which either symmetry or induction (or both) can be used solve the above-mentioned limitations. We give an example problem for each class. The classification is not exhaustive, but represents those problem types for which we have good techniques and/or algorithms.

The paper is organised as follows. In the remainder of this section we give an informal overview of results for symmetry and induction. In section 2 we give some formal definitions and background. In section 3 we define the problem classification and give examples for each; in section 4, we survey known results and discuss how they fit into the classification. Conclusions are presented in section 5.

1.1 Overview of results

Symmetry One approach that has been investigated to avoid state-space problems is the use of symmetry to replace a state-graph with a *quotient structure* in which states are replaced by *orbit representatives* (see, for example [16]). However this approach is, in most cases, not practical, as finding orbit representatives is hard. Indeed, the *orbit problem* as it is known, is as hard as the *graph isomorphism problem* [17].

Symmetry can also be used to reduce the number of *cases* that need to be checked when verifying a set of parameterised properties of a (parameterised) system. This use of symmetry is very different from that described previously – in this case we are not concerned with the number of *states* to be visited but the number of *cases* to be checked.

Induction Model-checking alone does not allow us to generalise about results, the problem is undecidable [4]. However, in some specific cases the problem is solvable [8, 27, 31, 10].

If the processes are terminating then, under some circumstances, with time, a system P_n will degenerate to a system P_{n-1}. Then, a proof of (certain types of) property of all systems of size $n-1$, together with an inductive step, will imply the proof of all systems of size n. If, on the on the hand, the individual processes within a given system are non-terminating, then the identification of an *invariant* process (or similar) [47, 59] is required. An invariant process is one which is *greater* (with respect to some suitable preorder relation) than any of the systems P_i. Under some circumstances, proof that the invariant process satisfies a temporal property implies that any system P_i satisfies the property.

Again, we have two very different techniques which rely on the same mathematical concept (induction this time), but which have a different effect, according to the context in which they are applied.

In this paper, we attempt to formalise each of the contexts in which symmetry and/or induction can be applied and to survey and classify recent results in this area.

2 Definitions

The systems of interest consist of the parallel composition of a number of processes, thus:

$$P_k = q||p_1||p_2||\ldots||p_k$$

for processes q and $p_1, p_2, \ldots, p_k$. The q and p_i are non-terminating (but finite-state), unless otherwise stated.

More specifically, we consider networks of *parameterised* processes, consisting of single process q (an *environment* process) together with k instantiations of the same, paramaterised process p, thus:

Definition 1. *An m-parameterised network of processes of size k consists of a process q in parallel with k parallel instantiations of a process $p = p(X_0, X_1, \ldots, X_m)$ with domains $D_0, D_1, \ldots, D_m$, thus:*

$$P_k = q||p(x_{10}, x_{11}, \ldots, x_{1m})||p(x_{20}, x_{21}, \ldots, x_{2m})||\ldots||p(x_{k0}, x_{k1}, \ldots, x_{km})$$

where $x_{ij} \in D_j$ for all i, j, $1 \leq i \leq k$ and $0 \leq j \leq m$.

Note that there is often no environment process q, in which case it is simply omitted from the above definition.

In many cases, we will be concerned with networks in which each of the instantiations of process p are identical up to renaming (that is, they are isomorphic). In this framework, we represent that situation by $m = 0$, $x_{i0} = i$, for $1 \leq i \leq k$ and so $P_k = q||p(1)||p(2)||\ldots||p(k)$ (or simply $P_k = p(1)||p(2)||\ldots||p(k)$ if there is no process q).

Temporal specifications (or *properties*) are expressed in temporal logic [24]. The logic used is usually either *propositional linear temporal logic* ($PLTL$ or PTL) [52], usually referred to as simply LTL, the branching time logic *Computation Tree logic* (CTL) or the more expressive CTL^*, and its *indexed* form ($ICTL^*$) [15, 14].

Definition 2. *A logic formula ϕ containing free variables indexed by $\alpha_0 \ldots \alpha_n$, written $\phi[\alpha_0 \ldots \alpha_n]$, is said to depend on $\alpha_0 \ldots \alpha_n$.*

The semantics of a network of processes is given by a Kripke structure, generated in the usual way from the interleaving product of the semantics of the individual processes.

Definition 3. *Let AP be a set of atomic propositions. A Kripke structure over AP is a triple $M = (S, R, L)$ where*

- *S is a finite set of states*
- *$R \subseteq S \times S$ is a transition relation, which must be total (i.e. for every state s_1 there exists a state s_2 such that $(s_1, s_2) \in R$).*
- *$L : S \to 2^{AP}$ is a labeling function which associates with each state a set of atomic propositions that are true in the state.*

Because the transition relation of a Kripke structure is always total, the relation R must be extended if some state s has no successor (that is, s is a *terminal state*). In this case R is modified so that $R(s,s)$ holds.

The traditional model-checking problem can be stated as follows:

`Given a Kripke Structure` $M = (S, R, L)$`, a state` $s \in S$`, and a temporal specification` ϕ`, is it true that` $M, s \models \phi$`?`

The verification problem for an infinite family of networks of process can be stated as follows:

`Given an infinite family` $F = \{P_i\}_{i=1}^{\infty}$ `of networks of processes` P_i `and a temporal specification` ϕ `is it true that that all the Kripke systems representing members of the family` F `satisfy` ϕ`?`

We now give some definitions which concern the underlying symmetry of Kripke structures.

Definition 4. *Given a Kripke structure* $M = (S, R, L)$*,*

1. *The automorphism group Aut* M *of* M *is the group consisting of all permutations acting on the state set* S *that preserve the transition relation* R*.*
2. *A symmetry group is any subgroup of Aut* M*.*
3. *Any symmetry group* G *acting on the state set* S *partitions the state set* S *into equivalence classes called orbits, where the orbit containing the state* s *is given by* $\theta(s) = \{t | (\exists \sigma \in G)(\sigma s = t)\}$*.*

Suppose we have a symmetry group acting on the states S of a Kripke structure as defined above, then from each orbit $\theta(s)$ we can pick a representative which we call $rep\theta(s)$. This representative can be any member of $\theta(s)$. A reduced (or *quotient*) Kripke structure can then be constructed by collapsing all the states in each orbit to a single representative state in each case.

Definition 5. *Given a Kripke structure* $M = (S, R, L)$ *with symmetry group* G*, a quotient model* $M_G = (S_G, R_G, L_G)$ *is the Kripke structure with state set* $S_G = \{\theta(s) | s \in S\}$*, the set of orbits of the states in* S*, transition relation* $R_G = \{(\theta(s_1), \theta(s_2)) | (s_1, s_2) \in R\}$*, and labelling function* L_G *given by* $L_G(\theta(s)) = L(rep(\theta(s)))$*.*

3 Scenarios and Examples

In this section we describe five scenarios in which symmetry and/or induction can be used to aid model-checking. In each case we give an example for which the approach is applicable.

3.1 State-space reduction

Goal: For constant k, show $M(P_k) \models \phi$, where P_k consists of k identical (up to renaming) copies of a process p, and ϕ may depend on the process identifiers

but is invariant over a given set of permutations.

Problem: the state-space is too large for traditional model-checking.

State-space reduction is achieved by model-checking over a *quotient structure* M_G (see definition 5) instead of the structure M which is too large. This is sound because if the property ϕ (and all atomic propositions contained thereof) is invariant under G, then $M, s \models \phi \iff M_G, \theta(s) \models \phi$ [12].

The applicability of this approach is limited for two reasons. First, finding a suitable symmetry group G is difficult (calculation of the entire automorphism group is as computatationally complex as graph isomorphism [17]). However if processes p_i, $i = 1, 2, \ldots, k$ are *isomorphic* then $Aut\ M = Aut\ CR$, where CR is the process communication graph for P_k, and $Aut\ CR$ is often relatively easy to compute.

Second, another restriction of this approach is the requirement that ϕ be invariant under G. In many cases this severely restricts the kinds of property that ϕ can express.

An example [30] that uses this approach is a solution to the mutual exlusion problem for a system of k identical processes. The mutual exclusion property states that

No two processes are ever in their critical section at the same time.

Assuming suitable propositions C_i (component i is in critical region), this is expressed in $ICTL$ as

$$AG(\wedge_{i \neq j} \neg(C_i \wedge C_j)).$$

This is a very simple case, not only are processes isomorphic but, since all processes can communicate with each other, the associated communication graph is the complete graph on n nodes, K_n. Since the automorphism group of K_n is the symmetric group of order n, (S_n), the automorphism group of the underlying Kripke structure M is S_n. As ϕ is invariant under S_n, by taking $G = S_n$, a quotient structure containing 2 states can be constructed. This structure is easily model-checked.

3.2 Case reduction

Goal: For constants k and r, $r \leq k$, for all distinct values of $t_1, t_2, \ldots, t_r$, where for all v, $1 \leq v \leq r$, $t_v \in \{1, 2, \ldots, k\}$, show that $M(P_k)_{t_1,t_2,\ldots,t_r} \models \phi[t_1, t_2, \ldots, t_r]$ where $M(P_k)_{t_1,t_2,\ldots,t_r}$ consists of k parallel instantiations p_i, $1 \leq i \leq k$, of the process $p = p(X_0, X_1, \ldots, X_m)$ such that

- $p_i = p(i, \hat{X}_{01}, \hat{X}_{02}, \ldots, \hat{X}_{0m})$, when $i \notin \{t_1, t_2, \ldots, t_r\}$,
- $p_i = p(i, \hat{X}_{v1}, \hat{X}_{v2}, \ldots, \hat{X}_{vm})$ otherwise

for fixed $\hat{X}_{l,j}$, $0 \leq l \leq r$, $1 \leq j \leq m$.

Problem: the cases generated by instantiating ϕ quickly suffer a combinatorial explosion.

We can often eliminate the need to consider all cases by appealing to symmetry. Consider the simplest case, where all processes are isomporphic and ϕ depends on process identifiers $t_1, t_2, \ldots, t_r$. Thus $P_k = p(1)||p(2)|| \ldots ||p(k)$. Then $M(P_k)_{t_1,t_2,\ldots,t_r}$ is identical to $M(P_k)_{t'_1,t'_2,\ldots,t'_r}$ up to renaming of processes. Suppose that $M(P_k)_{t_1,t_2,\ldots,t_r} \models \phi[t_1, t_2, \ldots, t_r]$. Any permutation σ taking $t_1, t_2, \ldots, t_r$ to $t'_1, t'_2, \ldots, t'_r$, will preserve the transition relation of the underlying Kripke structure (i.e. the underlying semantics). The two models will be bismilar and hence satisfy ϕ, provided it is suitably renamed under σ.

Namely, let $M = M(P_k)_{t_1,t_2,\ldots,t_r}$, and let s be a state, ϕ a CTL^* formula, and σ a permutation with M_σ, σs and $\sigma\phi$ the appropriate transformations under σ. (Hence, if $t_i\sigma = t'_i$, for $1 \leq i \leq r$, $M_\sigma = M(P_k)_{t'_1,t'_2,\ldots,t'_r}$.) Clearly there is a bisimulation between M and M_σ and so we have the following result: $M, s \models \phi \leftrightarrow M_\sigma, \sigma s \models \sigma\phi$. (The proof follows from the result that CTL^* is adequate with respect to bisimulation, established in [7]). This is similar to the result in section 3.1 but in this case, because we consider (unquotiented) structures, the formula itself can be transformed under a permutation σ as well as the model (often desirable).

This approach can be applied also when the processes p_i are not all isomorphic. Suppose that $p_2, p_3, \ldots, p_k$ are isomorphic to each other, but that p_1 is not isomorphic to the other processes. We therefore let q be p_1. (Note that the process q here should not be confused with a general environment process (see section 2), which need bear no relation to p.)

An example of this scenario is a communications service, with asynchronous communication [9]. Consider model-checking a system of $k-1$ basic call processes acting concurrently together with a basic call process augmented with additional feature behaviour. Thus $P_k = q(x_1)||p(x_2)|| \ldots ||p(x_k)$ where $(x_1, x_2, \ldots, x_k) = \sigma(1, 2, \ldots, k)$ for some permutation σ.

Suppose the feature is call forward unconditionally CFU (all calls to a subscriber are diverted to another user). A desirable property for all systems consisting of $k-1$ basic call processes plus one process with the additional CFU feature, (process j say, such that j forwards to l, $l \neq j$) has the form $\phi[i, j, l]$, where $\phi[i, j, l]$ is :

> *If user i rings user j then a connection between i and l will be attempted before user i hangs up.*

Assuming suitable variables *dialled* and *partner*, in LTL this is

$$[]((dialled[i] == j) \rightarrow ((partner[i] == chan\_name[l])\ \mathcal{P}\ (dev[i] == on)))$$

(where $\mathcal{P}$ denotes the temporal operator *precedes*).

We want to check property $\phi = \phi[i, j, l]$ for each associated model, for each set of parameters i,j and l. Each model $M(P_k)_{i,j,l}$ consists of $q(j)$ in parallel with the $k-1$ processes $p(u)$, $1 \leq u \leq k$, $u \neq j$. (Note that as processes i and l are basic call processes, the values of i and l do not effect the model.)

When k is only 4, potentially there are 48 cases to check (4 choices for j and i and 3 choices for l). However, applying the approach above, there are only 3 cases to check ($i = 0$, $j = 1$, $l = 2$; $i = 0$, $j = 1$, $l = 0$; and $i = 0$, $j = 1$, $l = 1$) – all other cases are just a permutation of one of these 3 cases.

3.3 Process degeneration

Goal: For all $k > k'$, where k' is a constant, show $M(P_k) \models \phi$, where ϕ does not depend on the process identifiers, P_k is m-parameterised with each p_i of the form $p(x_{i0}, x_{i1}, \ldots, x_{im})$ and the processes $p_1, p_2, \ldots p_k$ eventually terminate.

Problem: the state-space is too large for traditional model-checking.

Under certain circumstances, induction can be applied because once any process has *terminated* the system behaves like a system of $k - 1$ processes. (By termination we mean that the process is in a terminal state, see section 2.)

But, there are some constraints. First, ϕ cannot depend on processes identifiers, so it has to be a property like "a leader is eventually elected", or "a global variable z has an invariant property $z <= 10$". Second, the *actual* parameters of the p_i must fulfill certain properties, to ensure that there is a well-founded ordering on the $M(P_i)$. We call this constraint *feasability* and illustrate the concept below.

An example of this scenario is the following. During the tree identification phase of the IEEE 1394 High Performance Series Bus ("Firewire") protocol [41], processes within a given network choose a leader (that is a process which is parent to all of its neighbours) by sending a series of *be my parent* and *be my child* messages to their neighbours. Once a process has had a *be my parent* request accepted, it no longer participates in the protocol (that is it *terminates*). The protocol can be modelled as a system of processes each of which is a parameterised instance of a prescribed *Node* process [10]. The communication is again asynchronous. Consider such a system P_k. In this case $m = k$, $x_{i0} = i$, for $1 \leq i \leq k$ and $x_{ij} \in \{0, 1\}$ for all $1 \leq i, j \leq k$. We say that $X_{11} = x_{11}, X_{12} = x_{12}$, $\ldots, X_{kk} == x_{kk}$ is a *feasible assignment* of parameters $X_{11}, X_{12} \ldots, X_{kk}$ if $x_{ij} = x_{ji}$ and $x(ii) = 0$ for all $1 \leq i, j \leq k$, that is if the matrix A, where $A(i, j) = x_{ij}$ is an adjacency matrix of a graph. Similarly such an assignment is an *acyclic feasible assignment* if, further, such assignment of parameters corresponds to the adjacency matrix of an acyclic graph (i.e. a tree). A system P_k is called an AF (acyclic feasible) system of order k if it has an acyclic feasible assignment of parameters.

One correctness property ϕ relating to an AF system of Node processes is: is

A leader will always be chosen.

Assuming a suitable global variable *elected* to denote the process id of the elected leader, in LTL this is expressed by

$$[] <> (1 \leq elected \leq k).$$

Suppose that it is possible to check that ϕ holds for all AF systems P_i (of Node processes) of order $i_0 \leq i \leq k'$, for a constants i_0 and k'. Clearly if we can show that, for any $k > k'$, $M(P_{k-1}) \models \phi$ implies $M(P_k) \models \phi$ (that is, if we can prove an induction step), then it will follow that $M(P_i) \models \phi$ for all AF systems P_i, $i \geq i_0$. The proof of the induction step is as follows. If P_k is any AF system of order k then, as the associated communication graph is acyclic, at least one Node process must have only one neighbour (call these processes "leaf nodes"). One of the leaf nodes will be the first process to have its "be my parent" request accepted, and will no longer take part in communication. The system will now behave like an AF system of order $k-1$. Thus, as $M(P_{k-1}) \models \phi$, ϕ holds for P_k.

3.4 Simple induction

Goal: Show $\forall n.M(P_n) \models \phi$, where, each system P_k, consists of a (possibly null) process q together with n identical (up to renaming) copies of a process p, (p possibly identical to q) and ϕ does not depend on the process identifiers.

Problem: the problem is undecidable, using model-checking techniques. It is solvable, with suitable preorder and invariant.

There are several different results for this case; the common idea can be summarised as follows. Define a preorder $\preceq$ (a bisimulation in some approaches) and invariant $\mathcal{I}$ such that $M(P_n) \preceq \mathcal{I}$ for all P_n. The composition operator $||$ must be monotonic wrt $\preceq$ (for structures Q, Q', R and R', if $Q \preceq Q'$ and $R \preceq R'$, then $Q||R \preceq Q'||R'$.

The *invariant rule* for families of identical processes when there is no environment (i.e. q is null) is: for preorder $\preceq$ and invariant $\mathcal{I}$, if $\mathcal{I} \succeq M(p)$ and $\mathcal{I} \succeq \mathcal{I}||M(p)$ then $\mathcal{I} \succeq M(P_n)$ for all $n \geq 1$. If q is non-null, then the invariant is $q||\mathcal{I}$.

The preorder preserves ϕ so that $\mathcal{I} \models \phi \Rightarrow \forall n.M(P_n) \models \phi$. It therefore remains to check $Inv \models \phi$ using traditional model-checking techniques. The main difficulty (assuming we have established a suitable theorem for the preservation of ϕ wrt $\preceq$) is deriving a general method for constructing the invariant, an *abstraction* of the models in the infinite family.

A novel idea for constructing the invariant [19] is the following. Define a regular language to express the state propositions in ϕ and then uses an equivalence relation derived from the associated automaton to define the abstraction. Thus, the property "determines" the form of the abstraction.

Consider the following example: each process p_i has an associated bit variable x_i which it flips between 0 and 1. So, for any P_k, each transition represents an alternation of a variable x_i, for $1 \leq i \leq k$. A desirable property is that one can always reach a state in which at least two variables are 1:

For some i,j. it is possible that $x_i == x_j == 1$.

For any $M(P_i)$, the states we are interested can be encoded as strings $x_1 x_2 \ldots x_i$; so for the infinite family the language is

$$\{0,1\}^* 1 \{0,1\}^* 1 \{0,1\}^*.$$

From the associated automaton we derive a equivalence relation which partitions states into three classes: effectively, the state containing no 1's, the state containing one 1, and the state containing two 1s. The invariant $\mathcal{I}$ therefore contains these three classes, with appropriate transitions (i.e. transitions between the first two classes, and between the second two classes). Using model-checking we can show that the property holds for $\mathcal{I}$ (from any state we can reach a state with two 1's) and so we conclude that it holds for all such networks.

In [19], an example is given using synchronous Moore machines, we are not aware of any extension to asynchronous models of computation.

3.5 General induction

Goal: For all n, for fixed r_n, where $r_n \leq n$, for all distinct values of $t_1, t_2, \ldots, t_{r_n}$, where, for all v, $1 \leq v \leq r$, $t_v \in \{1, 2, \ldots, n\}$, show that $M(P_n)_{t_1,t_2,\ldots,t_{r_n}} \models \phi[t_1, t_2, \ldots, t_{r_n}]$ where P_n is m_n-parameterised with each p_i of the form $p_i = p(x_{i0}, x_{i1}, \ldots, x_{im_n})$.

Problem: the problem is undecidable, using model-checking, but it may be solvable with suitable preorder, invariant, feasability constraints and symmetry groups.

This general case combines the constraints and techniques of cases 3.2,3.3 and 3.4 in the following way. From case 3.2, we use symmetry groups for case explosion (because ϕ has form $\phi[t_1, t_2, \ldots, t_r]$), from case 3.3, we employ the notion of *feasible assignments* of parameters to the processes p_i, and from case 3.4, we construct an invariant necessary for the induction.

Examples of this scenario are numerous. For example consider the generalisation of the communications service of section 3.2, a network of fixed size (i.e. $k = 4$). Namely, we require to show that the result holds for any network of size of size n ($n \geq 3$), provided the actual parameters of the individual processes are restricted to feasible assignments. The feasability requirement is similar to that of section 3.3, namely, the communication topology of each P_i has to be "similar". In this example, the channels must be instantiated in the appropriate way so as to ensure peer to peer communication (hence m_n is a function of n). We have yet to define a more general notion of feasibility and to quantify the families of processes to which it applies. This is further work.

4 Survey

A useful survey of results in symmetry and induction in model-checking (up to 1995) is given in [20]. In the past few years there has been a wealth of publications; we attempt to summarise the significance of some of these here.

State-space reduction The basic idea of exploiting symmetries to reduce the size of state-spaces is first investigated within the context of Petri Nets [40, 57]. Ip and Dill [45] apply these ideas to the automatic verification of finite-state systems and introduce a new data type, *scalarset*, to the Murϕ protocol description language to imply symmetries within a state-space.

Clarke et al [16] and Emerson and Sistla [30] discuss symmetry reduction for systems when the transition relation is given either explicitly in terms of states or symbolically as a BDD. (Their results are summarised in [20]). The complexity of the orbit problem and examples of groups where the orbit problem is easy to solve are considered in [13].

These early results are extended to incorporate the notions of fairness [22] and partial-order reduction [25], and an on-the-fly extension is introduced [36]. Symmetries in logical subformulas are exploited [2, 50] in LTL and CTL^* respectively.

Symmetry reductions have been implemented within existing model-checkers. For example the SMV system [51] includes a symmetry reduction package which uses the scalarset approach of Ip and Dill [45]. A similar approach has been applied in the development of SymmSpin [6], a symmetry reduction package for SPIN [38]. More recently an extension of this approach, which exploits heap symmetries [43] have been implemented in dSPIN (a dynamic extension of SPIN) [42]. A reduction that exploits the symmetry induced when exploring state-spaces of concurrent software applications [34] is implemented within the Verisoft (state-exploration) tool [33]. The SMC (symmetry-based model checker) [56] is a model-checker that has been specifically designed to exploit both process symmetry and state symmetry when checking certain properties under different fairness assumptions.

Some approaches consider the case where systems are not totally symmetric, that is when they are *nearly symmetric* [23] or *partially symmetric* [37]. The case in which a property is not invariant under a given symmetry is investigated in [55].

Case Reduction While we are not aware of any examples of this application of symmetry in model-checking, a similar phenomena is explored in the isomorph-free model generation of Jackson, Jha and Damon [46]. They consider the problem of generation (by variable assignments) of models for checking properties of relational specifications, and reduce the number of models considered by partitioning the space into the equivalence classes of symmetrical interpretations.

Process degeneration Induction based on process degeneration has been investigated in [11], we are not aware of other related work.

Simple induction There are numerous results for this case. Early results involve families of systems consisting of identical processes. Kurshan et al [47] prove a a structural induction theorem for processes using the simulation

pre-order to generate the invariant. Similar results are achieved [8, 59] by establishing a bisimulation equivalence between global state graphs of systems of different sizes. Context-free network grammars are used [53, 19] to generate both generate suitable infinite families and to generate an invariant. In [27] it is shown that, for rings of token-passing processes, there exists a k such that the correctness of a ring with k processes implies the correctness of rings of arbitrary size.

Extensions to these early results, when a (non-trivial) environment process is involved, include [18, 31, 32, 5, 47, 1].

In [44] techniques are presented to automate the construction of abstractions of systems of identical components. An extension of this abstraction technique is implemented within the Murϕ verification system [21]. Similarly a fully automated approach for verifying parameterized networks with synchronous communucation is proposed in [28, 29], and a tool based on the network grammar approach [49] is designed to help in the construction of invariants.

Lack of space prohibits description of other related results which concentrate on the abstraction of a complex system, for example [35, 58].

General induction To our knowledge, this class of induction has not been investigated previously in its most general form. However, a relevant result concerns the verification of certain properties of families of systems comprising of multiple heterogeneous classes of processes [26]. In this case each class is instantiated by many homogeneous copies of a class *template* and model-checking for systems of arbitrary size n can be reduced to model-checking systems of size up to a small *cutoff* size.

5 Conclusions

Several limitations of the verification of networks of processes by model-checking are discussed. We bring together the numerous results in symmetry and induction in one coherent framework and propose five classes of problem in which either symmetry or induction (or both) can be used solve the limitations of state-space explosion, case explosion, or generalisation. We give an example problem for each class. The classification is not exhaustive, but represents those problem types for which we have good techniques and/or algorithms. Recent results in the field are discussed and classified according to the proposed classification. We identify the concept of feasible assignments for parameterised processes and suggest that this is an area for further work.

References

1. P.A. Abdulla and B. Jonsson. On the existence of network invariants for verifying parameterized systems. In E-R. Olderog and B. Steffen, eds. *Correct System Design, Recent Insight and Advances*, vol. 1710 *LNCS*, pp. 180–197. Springer-Verlag, 1999.
2. K. Ajami, S. Haddad, and J.M. Ilie. Exploiting symmetry in linear time temporal logic model check ing: One step beyond. In B.Steffen, ed. *Proc. TACAS '98*, vol. 1384 *LNCS*, pp. 52–67, 1998. Springer-Verlag.

3. R. Alur and T.A. Henzinger, eds. *Proc. CAV '96*, vol. 1102 *LNCS* 1996. Springer-Verlag.
4. K.R. Apt and D.C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.
5. P.C. Attie and E. A. Emerson. Synthesis of concurrent systems with many similar processes. *ACM Transactions on Programming Languages and Systems*, 20(1):51–115, January 1998.
6. D. Bosnacki, D. Dams, and L. Holenderski. A heuristic for symmetry reductions with scalarsets. In J.N Oliveira and P. Zave, eds. *Proc. FME2001*, vol. 2021 *LNCS*, pp. 518–533, 2001. Springer-Verlag.
7. M. Browne, E. Clarke, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1989.
8. M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81:13–31, 1989.
9. M Calder and A Miller. Using SPIN for feature interaction analysis - a case study. In M.B. Dwyer, ed. *Proc. SPIN 2001*, vol. 2057 *LNCS*, pp. 143–162, 2001. Springer-Verlag.
10. M Calder and A Miller. Using SPIN to analyse the firewire protocol – a case study. In S. Maharaj, J. Romijn, and C. Shankland, eds. *Proc. of the IEEE 1394 (FireWire) Workshop*, pp. 9–13, 2001.
11. M Calder and A Miller. Using SPIN to Analyse the Tree Identification phase of the IEEE 1394 High Performance Serial Bus (FireWire) protocol. *Submitted For Publication.*
12. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.
13. E.M. Clarke, E.A. E.M. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model-checking. In *[39]*, pp. 147–158, 1998.
14. E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Proc. of the Workshop in Logic of Programs*, vol. 131 *LNCS*, 1981. Springer-Verlag.
15. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
16. E.M. Clarke, R. Enders, T. Filkhorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1–2):77–104, 1996.
17. E.M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In C. Courcoubetis, ed. *Proc. CAV '93*, vol. 697 *LNCS*, pp. 450–461, 1993. Springer-Verlag.
18. E.M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking algorithms. In *Proc. PODC '87*, pp. 294–303, 1987. ACM Press.
19. E.M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In I. Lee and S.A. Smolka, eds. *Proc. CONCUR '95*, vol. 962 *LNCS*, pp. 395–407, 1995. Springer-Verlag.
20. E.M. Clarke and S. Jha. Symmetry and induction in model checking. In J. van Leeuwen, ed. *Computer Science Today: Recent Trends and Developments*, vol. 1000 *LNCS*, pp. 455–470. Springer-Verlag, 1995.
21. D. L. Dill. The murϕ verification system. In *[3]*, pp. 390–393, 1996.
22. E. A. Emerson and A. P. Sistla. Utilizing symmetry when model-checking under fairness assumptions: An automata-theoretic approach. *ACM Trans. on Programming Languages and Systems*, 19(4):617–638, July 1997.
23. E. Allan Emerson and Richard J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In L. Pierre and T. Kropf, eds. *Proc. CHARME '99*, vol. 1703 *LNCS*, pp. 142–156, 1999. Springer-Verlag.
24. E. Allen Emerson. Temporal and modal logic. In J. Van Leeuwen, ed. *Handbook Of Theoretical Science*, vol. B, chapter 16, pp. 995–1072. Elsevier Science Publishers B.V, 1990.
25. E. Allen Emerson, S. Jha, and D. Peled. Combining partial order and symmetry reductions. In E. Brinksma, ed. *Proc. TACAS '97*, vol. 1217 *LNCS*, pp. 19–34, 1997. Springer-Verlag.
26. E. Allen Emerson and V. Kahlon. Reducing model checking of the many to the few. In D.A. McAllester, ed. *Proc. CADE 2000*, vol. 1831 *LNCS*, pp. 236–254, 2000. Springer-Verlag.

27. E. Allen Emerson and K.S. Namjoshi. Reasoning about rings. In *Proc. POPL '95)*, pp. 85–94, 1995. ACM Press.
28. E. A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems (extended abstract). In *[3]*, pp. 87–98, 1996.
29. E. A. Emerson and K.S. Namjoshi. Verification of a paramaterized bus arbitration protocol. In *[39]*, pp. 452–463, 1998.
30. E. A. Emerson and A.P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1–2):105–131, August 1996.
31. S.M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992.
32. M.Girkar and R. Moll. New results on the analysis of concurrent systems with an indefinite number of processes. In B. Jonsson and J. Parrow, eds. *Proc. CONCUR '94*, vol. 836 *LNCS*, pp. 65–80, 1994. Springer-Verlag.
33. P. Godefroid. Model checking for programming languages using verisoft. In *Proc. POPL '97*, pp. 174–186, 1997. ACM Press.
34. P. Godefroid. Exploiting symmetry when model-checking software (extended abstract). In J. Wu, S. Chanson, and Q. Gao, eds. *Proc. FORTE/PSTV '99*, vol. 156 *IFIP*, pp. 257–275, 1999. Kluwer.
35. S. Graf. Verification of a distributed cache memory by using abstractions. In D.L. Dill, ed. *Proc. CAV '94*, vol. 818 *LNCS*, pp. 207–219, 1994. Springer-Verlag.
36. V. Gyuris and A.P. Sistla. On-the-fly model checking under fairness that exploits symmetry. *Formal Methods in System Design*, 15(3):217–238, November 1999.
37. S. Haddad, J-M Ilie, and K. Ajami. A model checking method for partially symmetric systems. In *Proc. FORTE/PSTV 2000*, 2000.
38. G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
39. A.J. Hu and M.Y. Vardi, eds. *Proc. CAV '98*, vol. 1427 *LNCS*, 1998. Springer-Verlag.
40. P. Huber, A.M. Jenson, L.O. Jepson, and K. Jenson. Towards reachability trees for high-level petri nets. In G. Rozenberg, ed. *Advances on Petri Nets*, vol. 188 *LNCS*, pp. 215–233, Springer-Verlag, 1984.
41. IEEE 1394-1995. *IEEE Standard for a High Performance Serial Bus Std 1394-1995*. Institute of Electrical and Electronic Engineers, August 1995.
42. R. Iosif and R. Sisto. dSPIN: a dynamic extension of SPIN. In D. Dams, R. Gerth, S. Leue, and M. Massink, eds. *Theoretical and Practical Aspects of SPIN Model Checking: Proc. SPIN '99*, vol. 1680 *LNCS*, pp. 20–33, Springer-Verlag, 1999.
43. R. Iosif. Symmetry reduction criteria for software model checking. To appear in *Proc. SPIN 2002*.
44. C. Norris Ip and David L. Dill. Verifying systems with replicated components in murϕ. *Formal Methods in System Design*, 14:273–310, 1999.
45. C.Norris Ip and D. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.
46. D. Jackson, S. Jha, and C.A. Damon. Isomorph-free model enumeration: a new method for checking relational specifications. *ACM Transactions on Programming Languages and Systems*, 20(2):302–343, March 1998.
47. R. P. Kurshan and K.L. McMillan. A structural induction theorem for processes. In *Proc. PODC '89*, pp. 239–247, 1989. ACM Press.
48. R.P. Kurshan, M. Merritt, A. Orda, and S.R. Sachs. A structural linearization principle for processes. *Formal Methods in System Design*, 5(3):227–244, December 1994.
49. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized networks of processes. *Theoretical Computer Science*, 256(1–2):113–144, April 2001.
50. G.S. Manku, R. Hojati, and R. Brayton. Structural symmetry and model checking. In *[39]*, pp. 159–171, 1998.
51. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
52. A. Pnuelli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
53. Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In *[54]*, pp. 151–165, 1989.

54. J. Sifakis, ed. *Proc. of the International Workshop in Automatic Verification Methods for Finite State Systems*, vol. 407 *LNCS*, 1989. Springer-Verlag.
55. A. P. Sistla and P. Godfroid. Symmetry and reduced symmetry in model checking. In G. Berry, H. Comon, and A. Finkel, eds. *Proc. CAV 2001*, vol. 2102 *LNCS*, pp. 91–103, 2001. Springer-Verlag.
56. A. P. Sistla, V. Gyuris, and E. A. Emerson. Smc: A symmetry-based model checker for verification of safety and liveness properties. *ACM Transactions on Software Engineering and Methodology*, 9:133–166, 2000.
57. P. H. Starke. Reachability analysis of petri nets using symmetries. *Systems Analysis–Modelling–Simulation*, 8(4/5):293–303, 1991.
58. F. Wang. Automatic verification of pointer data-structure systems for all numbers of processes. In J.M. Wing, J. Woodcock, and J. Davies, eds. *Proc. FM'99*, vol. 1708 *LNCS*, pp. 328–347, 1999. Springer-Verlag.
59. P. Wolper and V. Lovinfosse. Properties of large sets of processes with network invariants (extended abstract). In *[54]*, pp. 68–80, 1989.