Isomorphism Checking for Symmetry Reduction

Arend Rensink
Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE Enschede, The Netherlands

April 12, 2010

Abstract

In this paper, we show how isomorphism checking can be used as an effective technique for symmetry reduction. Reduced state spaces are equivalent to the original ones under a strong notion of bisimilarity which preserves the multiplicity of outgoing transitions, and therefore also preserves stochastic temporal logics. We have implemented this in a setting where states are arbitrary graphs. Since no efficiently computable canonical representation is known for arbitrary graphs modulo isomorphism, we define an isomorphism-predicting hash function on the basis of an existing partition refinement algorithm. As an example, we report a factorial state space reduction on a model of an ad-hoc network connectivity protocol.

1 Introduction

The core activity of any model checker is state space exploration; the most important problem encountered is the combinatorial state space explosion, due to which explicit-state model checking can become intractable even for small problem instances. One of the main methods to counter this is *symmetry reduction*, which is the principle of collapsing all states that are sufficiently similar to one single state. Under a well-chosen notion of "sufficiently similar", the reduced (*quotient*) state space is as good as the original one for checking the properties one is interested in.

In the original conception of, e.g., [5, 12], the notion of "sufficiently similar" is captured by symmetries computed on the level of the state space. That is, states are collapsed only if there are automorphisms of the overall transition system that map them onto one another — or in other words, if they are in the same orbit of the automorphism group. Let us call this relation *global symmetry*. However, the core property that causes the reduced state space to be as good as the original one is that it is *bisimilar* to that original, and for this purpose it is not necessary that collapsed states are globally symmetric: in fact, any congruence (or auto-bisimulation) will do. This was in fact clear from the beginning of the research in this area; some proposals to use a weaker congruence than global symmetry can be found in [9, 10]. Here, the authors propose to use *virtual*, respectively *local symmetry*, which is based on the structure of the states. This is the approach we follow as well.

We are interested in model checking software systems, both at design time (where the models are typically given in some visual language) and at implemen-

tation time (where the models are derived from code). To capture this domain in sufficient generality, for the purpose of this paper we assume system states to be arbitrary graphs; and we call states (locally) symmetric if the graphs are isomorphic. This raises an immediate complexity concern: to quote from [10], "the general problem of computing symmetry equivalence classes [...] has no polynomial-time solution. In general this makes it too costly to perform during model checking." Though we do not wish to dispute this *per se* (the phrase "in general" makes the statement almost irrefutable in any case), we do want to show that there are cases where the reduction due to isomorphism outweighs the cost.

An important element in the feasibility of our approach is the fact that one of the existing (very successful) isomorphism checking algorithms, developed by McKay [15], can be used not only to answer the standard question: "Given two graphs G and H, is H isomorphic to G?" but also, equally efficiently, the more general question: "Given a graph G and a set of pairwise non-isomorphic graphs S, does S contain a graph H isomorphic to G?" Namely, this algorithm works on the basis of *invariants* computed for the individual graphs at hand, and these invariants can be used as hash keys in storing the graphs of S. Although it follows from the complexity of isomorphism checking that no known algorithm computes canonical invariants (i.e., which never give rise to hash collisions for non-isomorphic graphs) in polynomial time, we report a variant of McKay's algorithm that is almost collision-free in our experiments, and has a worst-time complexity of $O(mn \log n)$ (where m is the number of edges and n the number of nodes in the graph). The experiments have been carried out in GROOVE [18], which is a tool for graph transformation-based model checking.

Note that the concept of an invariant is crucially different from that of a canonical representative, as used in the existing symmetry reduction methods, in that graphs cannot be reconstructed from their invariants. Indeed, the ability to do so would imply injectivity (and hence collision freedom) of the hash function.

Another contribution is that we show our reduction to be correct with respect to a stronger version of bisimilarity, which also preserves the *number* of outgoing transitions, *à la* resource bisimulation (see [6]). This is useful because, given a suitable notion of transition rates, resource bisimilarity also preserves stochastic logics such as CS(R)L (see [1, 2]).

To show the feasibility of our approach, we show results on modelling an ad-hoc network connectivity protocol, based on peer sampling. We show that the symmetry reduction achieved is very close to the theoretical maximum.

The remainder of the paper is structured as follows. In Section 2 we recall the notions of symmetry reduction and isomorphism. In Section 3 we discuss the algorithm used for computing the invariants. In Section 4 we present our experimental results, and in Section 5 we wrap up.

2 Definitions

As usual, we use state-transition systems as behavioural models. In the following we assume a universe Lab of labels. For the sake of generality we allow labelled transitions as well as (set-)labelled states.

Definition 1 (transition systems) *A transition system is a tuple* $\langle Q, T, L, \iota \rangle$, *where* Q *is a set of states,* T *a set of transitions with implicit source and target functions* $src, tgt: T \to Q$ *and labelling function* $lab: T \to \mathsf{Lab}$, $L: Q \to \mathbf{2}^{\mathsf{Lab}}$ *a state labelling function, and* $\iota \in Q$ *the initial state.*

Note that, in contrast to what is usual, T is not an (indexed) binary relation over states: though every transition $t \in T$ defines a triple $(src(t), lab(t), tgt(t))$, in our models there may be more than one identically labelled transition between the same states. The rationale behind this is that we also intend to use our framework for models where the number of outgoing transitions is relevant, for instance because they have an associated *rate*, as in certain types of stochastic models; see, e.g., [1].

In fact, without giving detailed definitions we observe that temporal logics such as CTL* [8], modal logics such as the modal μ-calculus [14], and stochastic logics such as CSL[1] [1] and CSRL[2] [2] all have a natural semantics over transition systems. As usual, we express the semantics by writing $K \models \phi$, for ϕ some formula in one of these logics, if K satisfies the property ϕ.

We will now recall a variation on bisimulation introduced by [6].

Definition 2 (resource bisimilarity) *Given a transition system* K, resource bisimilarity *is the largest equivalence* $\sim \subseteq Q \times Q$ *such that for all* $s_1 \sim s_2$, *all* $a \in \mathsf{Lab}$ *and all* $X \in Q/\!\sim$: *(i)* $L(s_1) = L(s_2)$ *and (ii)*

$$|src^{-1}(s_1) \cap lab^{-1}(a) \cap tgt^{-1}(X)| = |src^{-1}(s_2) \cap lab^{-1}(a) \cap tgt^{-1}(X)| \ .$$

We also say that two transition systems K_1, K_2 *are bisimilar (denoted* $K_1 \sim K_2$*) if* $\iota_1 \sim \iota_2$ *in the disjoint union* $K_1 \uplus K_2$.

The first condition states that for every state s, every label a and every maximal set of bisimilar states X, the *number* of outgoing a-labelled transitions from s to some state in X is well-defined modulo resource bisimilarity of s. It is clear that this gives rise to a more discriminating relation than the standard notion of strong bisimilarity (which does not compare numbers of transitions but just the existence of transitions). The following is known (see, e.g., [20, 1, 3]):

Proposition 3 *For two transition systems* K_1, K_2, $K_1 \sim K_2$ *implies that* $K_1 \models \phi$ *iff* $K_2 \models \phi$ *for arbitrary properties* ϕ *in CTL**, *μ-calculus and CS(R)L.*

Graphs and isomorphism. We now recall the standard concepts of graphs and graph isomorphism. We use the same universe of labels as for transition systems; the context will ensure that no confusion arises.

Definition 4 (graphs)

1. *A* graph *is a tuple* $\langle V, E \rangle$ *with* V *a set of nodes and* E *a set of edges with associated source and target functions* $src, tgt: E \to V$ *and labelling function* $lab: E \to \mathsf{Lab}$.

[1] Continuous Stochastic Logic; this can be interpreted by associating a rate $\lambda_t \in \mathsf{Real}$ with every transition t as a function of its label $lab(t)$, and setting the rate between two states to the sum of all individual transition rates.

[2] Continuous Stochastic Reward Logic, an extension to CSL which can be interpreted by associating a reward ρ_q with every state q as a function of its properties $L(q)$.

2. *Given two graphs G, H, a* morphism $f: G \to H$ *is a pair of functions $f_V: V_G \to V_H$ and $f_E: E_G \to E_H$ such that $src_H \circ f_E = f_V \circ src_G$, $tgt_H \circ f_E = f_V \circ tgt_G$ and $lab_H \circ f_E = lab_G$. f is an* isomorphism *if f_V and f_E are bijective; we sometimes write $f : G \cong H$ to denote this. G and H are then called isomorphic, denoted $G \cong H$.*
3. *Given a graph G and a set of colours Y, a* G-colouring *is a pair of functions $(c_V: V_G \to Y, c_E: E_G \to Y)$.*

We will silently extend functions such as f_V and c pointwise to sets, and also use their inverse as functions from sets to sets. Moreover, we will manipulate pairs of functions as one, as in $c \circ f$ or c^{-1} where f is a morphism and c a colouring.

Clearly, transition systems are graphs with extra structure, but we will not use this analogy here. Let Graph denote the universe of graphs. We recall the following (see, e.g., [22]):

Observation 5 (complexity of isomorphism) *Given two graphs G, H, deciding $G \cong H$ is in NP with respect to $|V_G|$, but not known either to be in P or to be NP-complete; it is thought to be neither.*

In this paper, we study transition systems in which every state is essentially characterised by an associated graph.

Definition 6 (graph-based transition systems) *A transition system K is called* graph-based *if it is equipped with an injective function $g: Q \to \mathsf{Graph}$ associating a graph with every state, such that $g(q) \cong g(q')$ implies $q \sim q'$ for all $q, q' \in Q$.*

The underlying idea of graph-based transition systems is that all essential characteristics of a state are encoded in its associated graph, but the chosen node and edge identities of the graph (i.e., the sets $V_{g(q)}$ and $E_{g(q)}$) are irrelevant. We claim that any existing specification language with semantics defined in terms of transition systems is or can be formulated so as to give rise to graph-based transition systems. In this paper, we report an experiment using graph transformation, where transitions are essentially transformation rule derivations, which are always well-defined modulo graph isomorphism. In the remainder of this paper, we work only with graph-based transition systems.

Definition 7 (transition system quotient) *Given a transition system K, a* state representation *is a function $\alpha: Q \to Q$ such that $g(\alpha(q)) \cong g(q)$ for all $q \in Q$. α is called* canonical *if $g(q) \cong g(q')$ implies $\alpha(q) = \alpha(q')$.*

For all $q \in Q$ let $f_q: g(q) \to g(\alpha(q))$ be the isomorphism from q's graph to that of its representative. The quotient of K with respect to α is then defined by

$$K/\alpha \;=\; \langle \alpha(Q),\, \{(t, f_{tgt(t)}) \mid src(t) = \alpha(Q)\},\, L \restriction \alpha(Q),\, \alpha(\iota),\, g \restriction \alpha(Q) \rangle$$

with $src((t,f)) = \alpha(src(t))$, $tgt((t,f)) = \alpha(tgt(t))$, and $lab((t,f)) = lab(t)$. ($f \restriction X$ denotes the restriction of f to the domain X.)

Hence, a state representation selects states with isomorphic graphs. For any non-injective α and finite K, the quotient K/α is clearly smaller than K; this is in fact our notion of symmetry reduction. The best reduction is achieved by using a canonical α. The following is the core insight for the usefulness of symmetry reduction.

Proposition 8 (correctness of symmetry reduction) *If K is a transition system and α a state representation, then $K/\alpha \sim K$.*

Of course, what we want in practice is not to first construct a transition system and then reduce it to its quotient, but rather to generate the quotient straight away. This on-the-fly symmetry reduction relies on the following core steps to add to an existing transition system K a single transition t with $src(t) \in Q$:

```
1  if ∃q ∈ Q : ∃f: g(tgt(t)) ≅ g(q)   (i.e., f → g(tgt(t))g(q) is an isomor-
   phism)
2  then let T := T ∪ {(t, f)}
3  else let Q := Q ∪ tgt(t);
4       let T := T ∪ {(t, id)};
5  endif
```

In terms of Definition 7, the **then** branch corresponds to setting $\alpha(tgt(t)) = q$, whereas α is the identity on $tgt(t)$ in the **else** branch. (In the latter case, the L-function should be extended as well, but we ignore this here.)

Complexity-wise, clearly the interesting step is Line 1, which searches for a state (in the set of previously detected states Q) with a graph isomorphic to that of the target of t.

3 Membership Checking Modulo Isomorphism

From the discussion above, we know that the core problem is the following:

> *Given a set of graphs S and a graph G, return an isomorphism $f: G \to H$ for some $H \in S$, or report failure and return $S \cup \{G\}$.*

On the face of it, this would seem to be even harder than deciding isomorphism for two given graphs, but as we will see, this is not really the case. If an algorithm reports failure though there exists an isomorphism of the right kind, we call this a *false negative*. If (and only if) the algorithm never yields false negatives, the derived function α is canonical. Note that even for a non-canonical α the reduction is correct, though the size of the reduced state space will depend on the "quality" of the algorithm, in terms of its number of false negatives (the fewer false negatives, the smaller the reduced state space).

Our solution is based on the concept of an *isomorphism invariant*:

Definition 9 (invariants)

1. *Given a set of hash values X, an* invariant hash function $\mathcal{H}$ *is a function $f\mathcal{H}: \mathsf{Graph} \to X$ such that $G \cong H$ implies $\mathcal{H}(G) = \mathcal{H}(H)$. Again, $\mathcal{H}$ is called canonical if the inverse is also true, i.e., $\mathcal{H}(G) = \mathcal{H}(H)$ implies $G \cong H$.*
2. *An* invariant colouring function $\mathcal{C}$ *maps every graph G to a G-colouring $\mathcal{C}(G)$ (for the same sets of colours Y), such that $f : G \cong H$ implies $\mathcal{C}(G) = \mathcal{C}(H) \circ f$. $\mathcal{C}$ is called canonical if the inverse is also true.*

For a function $k: A \to B$, let us denote by $im(k)$ the multiset of values used as k-images: formally, $im(k): B \to \mathsf{Nat}$ is defined by $y \mapsto |k^{-1}(y)|$ for all $y \in B$. The following states that every invariant colouring $\mathcal{C}$ gives rise to an invariant hash function $\bar{\mathcal{C}} : G \mapsto (im(\mathcal{C}(G)_V), im(\mathcal{C}(G)_E))$.

Proposition 10 *For an invariant colouring $\mathcal{C}$, if $G \cong H$ then $\bar{\mathcal{C}}(G) = \bar{\mathcal{C}}(H)$. If $\mathcal{C}$ is canonical, then the inverse also holds.*

In the remainder of this paper we use integer values as hash values and colours, i.e., we assume $X = Y = \mathsf{Int}$, and we assume the existence of a label hash function $hash\colon \mathsf{Lab} \to \mathsf{Nat}$. Easy examples are:

- $\mathcal{H} : G \mapsto |E_G|$, yielding the number of edges in a graph;
- $\mathcal{C} : G \mapsto (c_V, c_E)$ where $c_V : v \mapsto |tgt_G^{-1}(v)|$ and $c_E : e \mapsto hash(lab_G(e))$.

From an invariant colouring function $\mathcal{C}$ we can easily derive an invariant hash function $\mathcal{H}_\mathcal{C}$, as follows:

$$\mathcal{H}_\mathcal{C}\colon G \;\mapsto\; \textstyle\sum_{v \in V_G} \mathcal{C}(G)_V(v) + \sum_{e \in E_G} \mathcal{C}(G)_E(e) \;\; . \tag{1}$$

Our solution to the problem stated above is based on an invariant colouring $\mathcal{C}$. The set S is stored as a hash map $\mathsf{Int} \to \mathbf{2}^{\mathsf{Graph}}$ such that $\mathcal{H}_\mathcal{C}(G) = n$ for all $G \in S(n)$. The algorithm then consists of the following steps:

```
1  let  B = S(H_C(G));
2  for all  H ∈ B  such that  C̄(H) = C̄(G)  do
3      if  ∃f : G ≅ H.f ⊆ C(H)^{-1} ∘ C(G)
4      then return  f
5      endif
6  endfor
7  let  S := S[n ↦ (B ∪ {G})];
8  report failure
```

Line 3 specifies a search for isomorphisms within the relational space $\mathcal{C}(H)^{-1} \circ \mathcal{C}(G)$ (to be interpreted pairwise). At first sight, it would seem that this is no improvement over the original problem. Note, however, that if $B = \emptyset$, or if $\bar{\mathcal{C}}(G) \neq \bar{\mathcal{C}}(H)$ for some candidate $H \in B$, then this search will cheaply fail (due to Proposition 10). Furthermore, if $\mathcal{C}(G)$ is injective (which implies that G has no automorphisms) then $\mathcal{C}(H)_V^{-1} \circ \mathcal{C}(G)_V$ and $\mathcal{C}(H)_E^{-1} \circ \mathcal{C}(G)_E$ are one-to-one, meaning that there is at most one candidate f. Also, if $\mathcal{C}$ is canonical then *any* morphism f with $f_V \subseteq \mathcal{C}(H)^{-1} \circ \mathcal{C}(G)$ is an isomorphism, making the search linear in $|V_G| + |E_G|$; and even for non-canonical $\mathcal{C}$, the number of "non-iso-morphisms" may be small. Finally, recall that false negatives do not destroy the correctness (though they harm the effectiveness) of the algorithm: therefore, one may weaken the test in Line 3 by not exhaustively searching for an isomorphism but "giving up" after trying out a certain number of candidates.

We will call a *false hash positive* any graph $H \in B$ for which no isomorphism is found, and a *false colouring positive* any $H \in B$ for which $\bar{\mathcal{C}}(G) = \bar{\mathcal{C}}(H)$ but no isomorphism is found. A false positive can be due to a non-canonical $\mathcal{C}$, or due to collisions in the computation of $\mathcal{H}_\mathcal{C}$. Obviously, the fewer false positives, the less time is wasted in the search of Line 3; hence the fraction of false positives is a measure of the quality of the invariant colouring $\mathcal{C}$.

Stable partitions We have now reduced the symmetry reduction problem to the problem of finding a good invariant colouring $\mathcal{C}$, which gives rise to few false positives and is efficiently computable. Preferably $\mathcal{C}$ should be canonical,

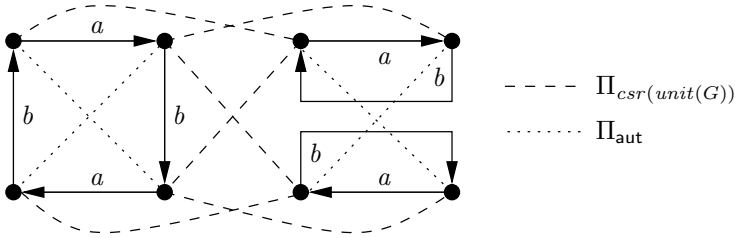

Figure 1: Graph with two partitions: The dashed lines connect the cells of $\Pi_{csr(\{V\})}$, the dotted lines do the same for Π_{aut}

but Observation 5 implies that there is little hope for any canonical $\mathcal{C}$ with polynomial worst time complexity.

We define our candidate colouring function by starting with the unit colouring *unit*, which assigns a single colour to all vertices and $hash(lab(e))$ to all edges e, and manipulating and refining this until it is close enough to canonical. In this process it is of supreme importance that all constructions $\mathcal{F}$ on the colourings *preserve invariance*, in the sense that if $c_G = c_H \circ f$ for colourings c_G and c_H and isomorphism $f: G \cong H$, then $\mathcal{F}(c_G) = \mathcal{F}(c_H) \circ f$. Since the unit colouring is clearly invariant, this ensures that the resulting colouring is, too.

Our construction is a combination of ideas from Paige and Tarjan [17] and McKay [15]. To explain this, first we have to recall some more concepts.

Given a graph G, a *partition* of G is a set of pairwise disjoint, non-empty sets $\Pi = \{V_1, \ldots, V_n\}$ such that $V_G = \bigcup \Pi$. For instance, any G-colouring c (using colours Y) gives rise to a partition $\Pi_c = \{\mathcal{C}(G)_V^{-1}(y) \mid y \in Y\}$. The *unit* partition is $\Pi_{unit} = \{\{V\}\}$. Partition Π_1 is *finer* than Π_2 (and Π_2 *coarser* than Π_1), denoted $\Pi_1 \leq \Pi_2$, if for all $V \in \Pi_1$ there is a $W \in \Pi_2$ such that $V \subseteq W$. Π is *stable* with respect to G (*equitable* in terms of [15]) if for all $V, W \in \Pi$, all $v_1, v_2 \in V$ and all $a \in \mathsf{Lab}$, the following conditions hold:[3]

$$\begin{aligned} |src^{-1}(v_1) \cap lab^{-1}(a) \cap tgt^{-1}(W)| &= |src^{-1}(v_2) \cap lab^{-1}(a) \cap tgt^{-1}(W)| \\ |tgt^{-1}(v_1) \cap lab^{-1}(a) \cap src^{-1}(W)| &= |tgt^{-1}(v_2) \cap lab^{-1}(a) \cap src^{-1}(W)| \ . \end{aligned}$$

We call a G-colouring c stable if Π_c is stable. For every partition Π of G, there is a unique *coarsest stable refinement* of Π, which we denote $csr(\Pi)$. One of the algorithms in [17] computes $csr(\Pi)$ from Π in time $O(m \log n)$, where $m = |E_G|$ and $n = |V_G|$. There is also a unique finest stable partition of every graph, induced by its automorphism group, defined by $\Pi_{\mathsf{aut}} = [v]_{\mathsf{aut}} v \in V_G$ where $[v]_{\mathsf{aut}} = \{f(v) \in V_G \mid f : G \cong G\}$. It follows that $\Pi_{\mathsf{aut}} \leq \Pi_{\mathcal{C}(G)}$ for any invariant colouring $\mathcal{C}$, and $\Pi_{\mathsf{aut}} = \Pi_{\mathcal{C}(G)}$ if $\mathcal{C}$ is canonical. For instance, Figure 1 shows the coarsest stable refinement of the unit partition $\{V\}$ as well as the finest stable partition Π_{aut}.

Phase 1: Refining the unit partition. The concepts of refinement and stability lift from partitions to colourings in the obvious way. From an arbitrary G-colouring c, we will attempt to construct a stable G-colouring $csr(c)$ such that $\Pi_{csr(c)} = csr(\Pi_c)$. (Note that this is not uniquely defined, since the colours are

[3]Note the analogy to resource bisimilarity (Definition 2): if we regard a transition system as a graph, then $\sim$ induces a "forward stable" partition, in which just the first of these conditions holds.

not fixed.) As discussed above, the construction should be invariant preserving. The following straightforward algorithm is our first candidate. It makes use of auxiliary hashing functions $h_E\colon \mathsf{Nat}^3 \to \mathsf{Nat}$ and $h_{src}, h_{tgt}\colon \mathsf{Nat} \to \mathsf{Nat}$

```
1  function rfn(c):
2     do
3        let c' := c;
4        let c := (c_V, c_E) with
5             c_E : e ↦ h_E(c'(src(e)), c'(tgt(e)), hash(lab(e)))
6             c_V : v ↦ ∑{h_src(c_E(e)) | v = src(e)}+∑{h_tgt(c_E(e)) | v = tgt(e)}
7     until |c'_V(V_G)| ≤ |c_V(V_G)|
8     return c';
```

Clearly, *rfn* is invariance preserving. The worst-time complexity is $O(md)$ where $m = |E_G|$ and d is the *diameter* of the graph (the maximum length of the shortest paths between connected nodes), which in the worst case equals $|V_G|$. Furthermore, if h_E, h_{src} and h_{tgt} are injective in all parameters, and h_{src} and h_{tgt} are such that the summation in the definition of c_V also never collides, then the colouring returned by the algorithm is a coarsest stable refinement of the input value. In practice we cannot make these guarantees, as we are working in a finite domain of numbers; in fact, it may happen that the output colouring does not even refine the input. Even in the case of collisions, though, the result is not worse, and very likely better, than the input, as it distinguishes at least as many elements. Our first candidate invariant colouring function is therefore $G \mapsto \mathit{rfn}(\mathit{unit}(G))$; below, we call this the "naive refinement".

A variation on Paige and Tarjan's partition refinement algorithm from [17] gives rise to another candidate colouring refinement function; for lack of space we cannot present it here, but let us call this function *pt*. Some care has to be taken that also this is invariance preserving; however, the advantage of this algorithm is that it has worst-case complexity $O(m \log n)$, where $n = |V_G|$, and is guaranteed to yield the coarsest stable refinement. Our second candidate invariant colouring is $G \mapsto \mathit{pt}(\mathit{unit}(G))$.

Phase 2: Breaking symmetries and summing up. As Figure 1 shows, not surprisingly the coarsest stable partition refining the unit partition is in general coarser than Π_{aut}. Inspired by McKay [15], we have implemented a further refinement based on the idea of *symmetry breaking*. Here, we successively change the colour of each of the vertices in every non-trivial cell of the input colouring's partition, and sum all the resulting colourings, according to the following algorithm (which uses an auxiliary hash function h_{brk}).

```
1  function breakSym_F(c):
2      let c := F(c);
3      for all v ∈ V_G with ∃v' ≠ v : c_V(v) = c_V(v') do
4          let c := c + F(c[v ↦ h_brk(c(v))])
5      endfor
6      return c
```

The parameter $\mathcal{F}$ is a transformation from G-colourings to G-colourings. For an arbitrary invariance-preserving transformation $\mathcal{F}$, $\mathit{breakSym}_{\mathcal{F}}$ is also

invariance-preserving. In particular, this is the case if we take $\mathcal{F}$ to be *rfn* defined above, or the Paige-Tarjan alternative. $breakSym_{\mathcal{F}}$ potentially refines its input further, because breaking the symmetry at elements of distinct cells of Π_{aut} will in general affect its rest of the graph differently. This gives rise to two improved invariant colouring functions, $G \mapsto breakSym_{rfn}(unit(G))$ and $G \mapsto breakSym_{pt}(unit(G))$.

4 Example: Ad-Hoc Network Connectivity

We illustrate the effectiveness of our method on the basis of an experiment previously reported in [7], where we modelled several versions of an ad-hoc network connectivity protocol originally proposed in [13]. This is an interesting case in that it concerns a realistic system in a very active application area, which, moreover, has a very high degree of non-trivial symmetry: all network nodes can be automorphic. It is therefore to be expected that isomorphism-based symmetry reduction can be large, whereas traditional symmetry reduction methods do not apply. Moreover, the properties checked on the model are formulated in CSRL, hence it is important that our reduction respects resource bisimilarity rather than standard bisimilarity (see Proposition 3).

The protocol. The system being modelled is an ad-hoc network, which consists of a set of linked network nodes but lacks a central server. Instead, the network nodes must themselves acquire and maintain knowledge of the structure of the network. This is done by so-called peer sampling, in which the nodes continuously exchange information about their "neighbours" or "peers" (the nodes they know about). The goal of this behaviour is to maintain a well-balanced network. In [13] it is assumed that each node knows only a small number of its peers; the set of peers known to a node is called its view, and has a maximum size. The behaviour of every node is to repeatedly execute the following steps:

1. The node becomes active and starts its peer sampling;
2. The node randomly selects a peer from its view;
3. The node and its peer send their view and/or receive a view in return;
4. The node and/or its peer merge the received view with their own view;
5. The node and/or its peer prune excess peers from their merged view;
6. The node is deactivated.

This sequence is taken to be atomic for each node, meaning that at most one node is active at any given time. We call the network *stable* if no node is active.

One of the parameters of the protocol is the communication policy in (step 3), which can be *push* (in which the node sends to its selected peer), *pull* (in which the peer sends to the original node) or *both* (push-pull). Other parameters are the view size and the network size. In the original presentation of [13] there are further parameters, but we can ignore them for the purpose of this paper.

A typical question that one can ask about this system is how well the network keeps up over time: for instance, can it fragment into disconnected parts, and if so, what is the probability of this happening, as a function of time? To be able to answer such questions, we have modelled the protocol in GROOVE (see [18]). The model consists of an initial state, shown in Figure 2, and a set of graph

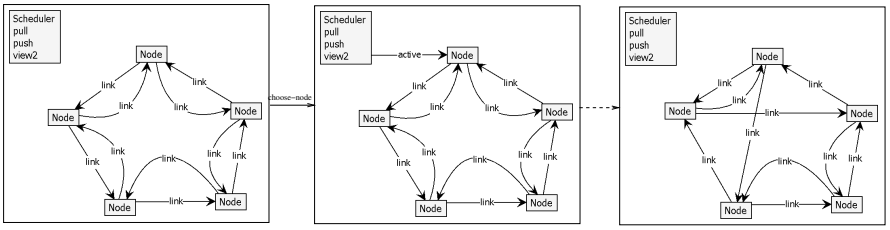

Figure 2: Initial network, node selection, and a stable next state.

transformation rules capturing the steps described above. This automatically results in a graph-based transition system in the sense of Definition 6.

Reduction. First we analyse the maximal achievable symmetry reduction, without taking the actual protocol into account. For a given network of size N and view size C, the number of possible configurations is $\binom{(N-1)}{C}^N$: each node has C distinct neighbours in its view, out of a possible $N-1$, and there are N nodes. This function clearly grows quite fast. It is less straightforward to compute the number of configurations modulo isomorphism, but a lower bound for that number can be established as follows. For every configuration graph, there are $N!/P$ distinct isomorphic graphs over the same set of nodes, where P is the size of the automorphism group of the graph. Namely, every permutation of the set of nodes gives rise to an isomorphic graph, but not all of those will be distinct; instead, each distinct isomorphic graph is obtained in this way through P different permutations. The size of the automorphism group is not easily predicted, but for larger random graphs its average is known to approach 1. Thus, we can expect a theoretical maximal reduction factor of at most $N!$, and for larger N the actual maximal reduction will probably be close to this.

To confirm this hypothesis, we have created another model in GROOVE to generate all the possible network configurations modulo isomorphism, for various network and view sizes. The results, as far as we have been able to compute them, are given in Table 3. It can be seen that, as expected, for larger networks the actual maximal reduction approaches the theoretical maximum quite closely.

Note that the numbers for given N and C are always equal to that for N and $N-C-1$. This is to be expected, since every configuration of view size C can be turned into its dual one of view size $N-C-1$ by defining links precisely where there are none in the original.

The *real* reduction depends in addition on the set of configurations that are reachable in the model. If the reachable configurations (without reduction) include few of the possible isomorphic representatives, the actual reduction factor may be much smaller than the theoretical maximum. However, this turns out not to be the case: the reachable configurations include almost all of the possible ones. Table 4 shows the size of the total state space (obtained from another model, specified in μCRL; see [7]) and the reduced state space, as well as the number of (reduced) reachable stable configurations. The reduction factors of the state space are of the same order of magnitude as in Table 3.

For the entries marked "–" in this table, we have been unable to fully explore the state space. The largest sizes we have been able to cope with are $N = 7, C = 2$ and $N = 7, C = 4$. Note that in all explored cases, the total state space size is in the order of 50 times as large as the number of reachable configurations.

C	N	4	5	6	7	8
	Max. reduction	*24*	*120*	*720*	*5,040*	*40,320*
1	**Normal**	81	1,024	1.6×10^4	2.8×10^5	5.8×10^6
	Reduced	6	13	40	100	291
	Reduction factor	*14*	*79*	*391*	*2,799*	*19,810*
	% of max	*56%*	*66%*	*54%*	*56%*	*49%*
2	**Normal**	81	7,776	1.0×10^6	1.7×10^8	3.8×10^{10}
	Reduced	6	79	1,499	35,317	9.7×10^5
	Reduction factor	*14*	*98*	*667*	*4,838*	*39,103*
	% of max	*56%*	*82%*	*93%*	*96%*	*97%*
3	**Normal**		1,024	1.0×10^6	1.3×10^9	2.3×10^{12}
	Reduced		13	1,499	2.7×10^5	–
	Reduction factor		*79*	*667*	*2,799*	–
	% of max		*66%*	*93%*	*99%*	–
4	**Normal**			15,625	1.7×10^8	2.3×10^{12}
	Reduced			40	35,317	–
	Reduction factor			*391*	*4,838*	–
	% of max			*54%*	*96%*	–
5	**Normal**				2.8×10^5	3.8×10^{10}
	Reduced				100	9.7×10^5
	Reduction factor				*2,799*	*39,103*
	% of Max				*56%*	*97%*

Table 3: Possible network configurations, normal and reduced (N = network size, C = view size).

We can therefore conjecture that the state space of the $N = 7, C = 3$ case is around 10 million; this is currently beyond the scope of GROOVE.

An interesting observation is that for view size $C = 2$ and network sizes $N = 6$ and $N = 7$, the number of reachable configurations is actually very slightly smaller than the number of possible configurations. When we observed this first, we suspected an error in the implementation, but a closer analysis explains this behaviour: in the *push* policy, any configuration *in which no node*

C	N	4	5	6	7
2	**Total state count**	945	1.2×10^5	1.9×10^7	–
	Reduced state count	80	1,850	56,843	1.5×10^6
	Reduced configurations	6	79	1,498	35,314
3	**Total state count**		321	2.3×10^7	–
	Reduced state count		321	56,843	–
	Reduced configurations		13	1,499	–
4	**Total state count**			3.9×10^5	–
	Reduced state count			1,247	1.7×10^6
	Reduced configurations			40	35,317
5	**Total state count**				–
	Reduced state count				4,555
	Reduced configurations				100

Table 4: Reachable states and (stable) configurations in the "push" protocol (N = network size, C = view size).

Refinement algorithm		**Naive**		**Paige-Tarjan**	
Symmetry breaking		No	Yes	No	Yes
Predicted isomorphisms		1,509,212	1,395,789	1,510,256	1,397,223
False pos	**Hash**	113,817	394	114,867	1,829
	% of predicted	*7.5%*	*0.0%*	*7.6%*	*0.1%*
	Colouring	113,423	0	113,005	0
	% of predicted	*7.5%*	*0.0%*	*7.5%*	*0.0%*
Time (ms)	**Total**	632,634	492,700	585,274	587,148
	Iso check	67,656	60,125	174,659	184,404
	% of total	*11%*	*12%*	*30%*	*31%*
	Hashing	28,572	41,491	124,415	153,536
	% of iso check	*42%*	*69%*	*71%*	*83%*
	Searching	23,000	1,412	20,818	1,481
	% of iso check	*34%*	*2%*	*12%*	*1%*

Table 5: Performance of the isomorphism checking algorithm in various versions, for $N = 7, C = 2$ and the push *policy*

shares a peer with any of its predecessors must be unreachable, unless it is the initial configuration. Such configurations can be constructed as soon as $N \geq 3C$.

Isomorphism checking. Apart from the achieved reduction, an interesting question is how well our algorithm for isomorphism checking performs. To be precise, we may ask the following questions:

1. How many false positives does the hash function $\mathcal{H}_\mathcal{C}$ give rise to?
2. How many false positives does the colouring function $\mathcal{C}$ give rise to?
3. How much time is spent in isomorphism checking?
4. How much of this time is spent calculating the colourings, and how much searching for an isomorphism within the remaining space?

These questions are answered in Table 5 for the following four cases: our naive partition refinement and the Paige-Tarjan algorithm, both with and without symmetry breaking. The results were obtained using a 64-bit JVM (version 1.5) running on a Intel Xeon X5355 (2.66GHz) with 20GB of memory.

An analysis of the results gives rise to the following observations:

- The very simple calculation of $\mathcal{H}_\mathcal{C}$ from $\mathcal{C}$, defined in (1), actually performs rather well: it gives rise to a negligible number of additional false positives (with respect to the false positives of $\mathcal{C}$ itself).
- The symmetry breaking phase during invariant colouring clearly pays off: the number of false positives of the colouring drops to zero, and the time spent searching for isomorphisms drops off correspondingly.
- The Paige-Tarjan algorithm, which has a better worst-time complexity than our naive implementation ($O(m \log n)$ versus $O(mn)$, where n is the number of nodes), nevertheless performs worse. We speculate that this is due to the fact that the algorithm has a rather large overhead, whereas the graphs in question are actually relatively small.

As an aside, we note that the absolute timing figures should be taken with a grain of salt. GROOVE is implemented in Java, which has unpredictable timing behaviour due to the built-in automatic garbage collection, and moreover, the results were obtained on a server shared with other users. Most of the time *not* spent in the isomorphism check is actually spent in graph matching, which is a major issue in graph transformation. Nevertheless, in [7] we have shown that (for this case study) GROOVE outperforms a distributed implementation of μCRL, both in performance and in the size of the models that can be explored.

Stochastic model checking. To the model described above we added probabilities, transition rates and reward measures, and we calculated long-run averages of the rewards. This analysis is cubic in the size of the state space, so the reported symmetry reduction is instrumental in scaling to larger models. In [7] we also speculated on the use of CSRL to model check further stochastic properties, such as "Does the configuration become disconnected with a certain probability after a certain amount of time?". Since our symmetry reduction preserves properties of this logic, it continues to be useful for that purpose.

5 Conclusion

Summarising, the contributions of this paper are the following:

- We propose to use symmetry reduction by collapsing isomorphic states. This has a greater potential for reduction than some of the existing methods but was thought to be prohibitively expensive.
- We have shown that this reduction preserves resource bisimilarity, and hence preserves all properties in stochastic logics (besides the usual temporal properties).
- We have described and implemented an algorithm for the detection of isomorphic representatives in a set of previously explored states, based on the concept of invariant colourings, combining and adapting two existing algorithms. Though the worst-time complexity is not polynomial, our experience is that for models encountered in practice this works well.
- We have shown the effectiveness of the symmetry reduction through a case study involving a model of an ad-hoc network peer sampling protocol.

Of course, it is dangerous to draw conclusions about the general effectiveness of the method based on a single experiment. However, earlier experiments reported in [19] on some other examples (including the standard dining philosophers problem, a mutual exclusion protocol and a concurrent list append function, but using only naive refinement and no symmetry breaking) showed comparable results. We conclude that isomorphism-based symmetry reduction can pay off, despite the sombre opinion expressed in [10] (see the introduction).

Related work. In the introduction we have already referred to the original work on symmetry reduction, as well as some of the follow-up. Let us stress again that, since we are interested in system properties that are preserved by (resource) bisimilarity, we are content with a relation that is weaker than global symmetry (relying on automorphisms of the entire state space) studied in, e.g.,

[5, 12]. Rather, for the purpose of this paper we concentrate on local symmetry and *assume* that this implies resource bisimilarity (through the concept of graph-based transition system, Definition 6) — observing that this indeed holds true for all specification formalisms we are aware of, provided that the underlying graphs encode all structural state information relevant to the behaviour. Thus, we actually avoid the question that is at the core of [5, 9], namely under what circumstances local symmetry implies global symmetry (in [5]) or bisimilarity (in [9]). Indeed, we do not have any claim as to optimality of the reduction, as in [9]).

The closest in aim to our work is by Iosif, Bosnacki and others in [4, 10, 11, 21], in the context of software model checking. The differences between these approaches and our work are several:

1. Their graphs are specialised to model processes and heaps, and the sorting criteria developed are dedicated to that structure. It is not easy to see how to encode the ad-hoc networks studied in Section 4 into their framework, without losing symmetries.
2. We always test for full isomorphism, rather than relying on heuristics. Though this is prohibitively expensive in the worst case, the experiment of Section 4 shows that this can pay off for nontrivial systems.
3. A disadvantage of our framework is that it does not give rise to canonical state representatives. Instead, we store the state space as a hash function, and apply the potentially expensive search algorithm of 6 to look up states.

Future work. To further improve our results, there is an alternative approach that we plan to try out: namely, to fully exploit the canonical labellings generated by McKay's algorithm in [15]. Compared to the invariant colourings of Definition 9.2, the canonical labelling $\mathcal{L}$ satisfies a weaker condition, viz. if $G \cong H$ then *there is* an isomorphism $f : G \cong H$ such that $\mathcal{L}(G) = \mathcal{L}(H) \circ f$, rather than this condition holding *for all* isomorphisms. However, $\mathcal{L}$ is canonical in the sense that the inverse implication also holds, and moreover, $\mathcal{L}(G)_V$ and $\mathcal{L}(G)_E$ are guaranteed to be injective — at the potential cost of exponential worst-time complexity, as must be the case due to Observation 5. However, [16] reports that "hard graphs" are rare, and we may conjecture that it is unlikely to find them in the systems we are modelling.

Using $\mathcal{L}$, we might be able to improve our results, because it then becomes possible to use canonical graph representatives as in other existing work on symmetry reduction, thus removing the disadvantage noted in item 3 above.

References

[1] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time markov chains by transient analysis. In E. A. Emerson and A. P. Sistla, eds., *CAV*, vol. 1855 of *LNCS*, pp. 358–372. Springer, 2000.

[2] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In Montanari, Rolim, and Welzl, eds., *ICALP*, vol. 1853 of *LNCS*, pp. 780–792. Springer, 2000.

[3] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] D. Bosnacki, D. Dams, and L. Holenderski. A heuristic for symmetry reductions with scalarsets. In J. N. Oliveira and P. Zave, eds., *FME*, vol. 2021 of *LNCS*, pp. 518–533. Springer, 2001.

[5] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[6] F. Corradini, R. D. Nicola, and A. Labella. Graded modalities and resource bisimulation. In C. P. Rangan, V. Raman, and R. Ramanujam, eds., *FSTTCS*, vol. 1738 of *LNCS*, pp. 381–393. Springer, 1999.

[7] P. Crouzen, J. C. van de Pol, and A. Rensink. Applying formal methods to gossiping networks with mCRL and Groove. *ACM SIGMETRICS Performance Evaluation Review*, 36(3):7–16, Dec. 2008.

[8] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[9] E. A. Emerson, J. Havlicek, and R. J. Trefler. Virtual symmetry reduction. In *LICS*, pp. 121–131, 2000.

[10] R. Iosif. Exploiting heap symmetries in explicit-state model checking of software. In *ASE*, pp. 254–261. IEEE Computer Society, 2001.

[11] R. Iosif. Symmetry reduction criteria for software model checking. In Bosnacki and Leue, eds., *SPIN*, vol. 2318 of *LNCS*, pp. 22–41. Springer, 2002.

[12] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1-2):41–75, 1996.

[13] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. Comp. Syst.*, 25(3), 2007.

[14] D. Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[15] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[16] B. D. McKay. *nauty User's Guide (Version 2.4)*, Oct. 2007. See http://cs.anu.edu.au/~bdm/nauty/.

[17] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[18] A. Rensink. The GROOVE simulator: A tool for state space generation. In J. Pfalz, M. Nagl, and B. Böhlen, eds., *AGTIVE*, vol. 3062 of *LNCS*, pp. 479–485. Springer, 2004.

[19] A. Rensink. Isomorphism checking in GROOVE. In Zündorf and Varró, eds., *GraBaTs*, vol. 1 of *Electr. Comm. of the EASST*, September 2007.

[20] C. Stirling. The joys of bisimulation. In L. Brim, J. Gruska, and J. Zlatuska, eds., *Mathematical Foundations of Computer Science (MFCS)*, vol. 1450 of *LNCS*, pp. 142–151. Springer, 1998.

[21] M. Veanes, J. P. Ernits, and C. Campbell. State isomorphism in model programs with abstract data structures. In J. Derrick and J. Vain, eds., *Formal Techniques for Networked and Distributed Systems (FORTE)*, vol. 4574 of *LNCS*, pp. 112–127. Springer, 2007.

[22] E. W. Weisstein. Isomorphic graphs. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/IsomorphicGraphs.html, 2002.