State Space c-Reductions of Concurrent Systems in Rewriting Logic*

Alberto Lluch Lafuente[1], José Meseguer[2], and Andrea Vandin[1]

[1] IMT Institute for Advanced Studies Lucca, Italy
[2] University of Illinois in Urbana-Champaign, USA

Abstract. We present *c-reductions*, a state space reduction technique. The rough idea is to exploit some equivalence relation on states (possibly capturing system regularities) that preserves behavioral properties, and explore the induced quotient system. This is done by means of a canonizer function, which maps each state into one (of the) canonical representative(s) of its equivalence class. The approach exploits the expressiveness of rewriting logic and its realization in Maude to enjoy several advantages over similar approaches: flexibility and simplicity in the definition of the reductions (supporting not only traditional symmetry reductions, but also name reuse and name abstraction); reasoning support for checking and proving correctness of the reductions; and automatization of the reduction infrastructure via Maude's meta-programming features. The approach has been validated over a set of representative case studies, exhibiting comparable results with respect to other tools.

1 Introduction

State space reduction techniques have been extensively investigated since the birth of automated verification and have contributed to its success by enhancing the performance of tools and allowing the analysis of larger and larger systems. State space reduction techniques have been particularly successful in model checking [1], where techniques such as abstract interpretation, partial order reduction, and symmetry reduction allow to consider smaller though equivalent systems and thus save precious space and time resources.

Symmetry reduction, for instance, enjoys a vast literature [2] but, while most of the authors agree on the effectiveness of the technique, symmetry reduction has not established itself as standard feature of verification tools as is the case of other state space reduction techniques. Notable examples are model checkers such as SPIN [3], where symmetry reduction and abstract interpretation are not built-in while other techniques like partial order reduction are.

There are several reasons for this: (i) automatic detection of system regularities is a hard task, very often delegated to the system designer since there are only

* Work supported by NSF Grant CCF 09-05584, AFOSR Grant FA8750-11-2-0084 and the EU Project ASCENS.

few solutions available (e.g. [4]); (ii) their exploitation is done by complementing (or enriching) the system description language with some meta-data in a different language, so that the user is hence required to use this new language; (iii) the implementation of state space reduction techniques has to be combined (both theoretically and practically) with the rest of the techniques and algorithms implemented in the model checker, and often this integration effort has to be repeated for every new version, improvement or technique of the model checker; and (iv) checking correctness of the defined reductions is not easy and requires reasoning techniques (e.g. theorem proving) that are not integrated in the model checking framework, or that are not part of the user's skills.

Our work started with the following question: *can we use rewriting logic to generalize symmetry reduction techniques in a way that does not suffer the above problems (i)–(iv) and, in addition, effectively improves the performance of state space exploration?* Our hope on the suitability of rewriting logic (and its realization in Maude [5]) for such purposes was motivated by the possibility of exploiting its expressiveness and formal verification support [6] (including model checking and theorem proving).

Contributions. This paper offers a first answer to the above question by presenting a generalization of symmetry reductions called c*-reductions*. Its key idea is the reduction of each state into one of the *canonical* representatives of an equivalence class induced by a relation on states that preserves the system properties under study (i.e. a bisimulation). The approach is fully general: it subsumes many reduction techniques (symmetry reduction, name reuse and name abstraction) and can be applied to any Kripke structure. However, the method is particularly beneficial when Kripke structures are described by rewrite theories (and any computable Kripke structure can be formally specified by a finitely described rewrite theory [7]).

The c-reduction approach tackles the above mentioned issues (i)–(iv) in the following way: (i) reductions are defined using ordinary ingredients of the system description language, namely functions and equations; (ii) the implementation of reductions does not interfere with the theory of rewriting logic or with the Maude engine and its commands, since it is essentially based on equational simplification, a standard feature of the setting; (iii) the approach is highly automatizable (thanks to Maude's metaprogramming facilities based on logical reflection) and, at the same time, flexible enough to allow customization (e.g. by allowing the user to define the canonization functions or parts of them); (iv) correctness checks are semi-automatized by techniques well supported by the Maude formal environment, e.g. critical pair analysis to check that canonizer functions do not "interfere" with other functions and are "coherent" with respect to behavioural rules.

We have evaluated our approach over an ample set of examples by considering the ease of defining reduction strategies, the effectiveness of the correctness checks, and the performance of the resulting reductions. With respect to previous works we have observed performance gains in some cases (including previous implementations of symmetry reductions in Maude [8]), more flexibility in the definition of reductions, which allow us to subsume a wide range of reductions including permutation and rotation symmetries, name reuse and name abstraction, which have interesting applications (e.g. implementation of the operational semantics of languages with dynamic features such as resource allocation). A preliminary version of our tool is available for download [9].

Synopsis. §2 offers the necessary background. §3 presents the `c`-reduction technique in a generic way, focusing on Kripke structures. §4 describes the realization of `c`-reductions in rewriting logic, highlighting the theoretical results, and the reasoning and verification mechanisms and tools, that can be exploited to analyze and develop and `c`-reductions. §5 evaluates the performance of `c`-reductions through some representative examples. Finally, §6 concludes the paper, describes subjects of current work and outlines future research directions.

2 Preliminaries

We focus on system specifications in the form of theories of rewriting logic [6], which can be realized in practice in terms of Maude modules [5].

Definition 1 (rewrite theory). *A* rewrite theory $\mathcal{R}$ *is a tuple* $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ *where* Σ *is a signature, specifying the basic syntax (function symbols) and type infrastructure (sorts, kinds and subsorting) for terms, i.e., state descriptions;* E *is a set of (possibly conditional) equations, which induce equivalence classes of terms, and (possibly conditional) membership predicates, which refine the typing information;* A *is a set of axioms which also induce equivalence classes of terms, i.e., equational axioms describing structural equivalences between terms, like associativity and commutativity;* R *is a set of (possibly conditional) non-equational rules, which specify the local concurrent transitions in a system whose states are* $E \cup A$*-equivalence classes of ground* Σ*-terms; and where* $\phi : \Sigma \to \mathcal{P}_{fin}(\mathbb{N})$ *is a* frozenness map*, assigning to each function symbol* f *of arity* n *a subset* $\phi(f) \subseteq \{1..n\}$ *of its frozen argument positions, i.e. positions under which rewriting with rules in* R *is forbidden.*

For the sake of simplicity, we assume that the system under study is described by a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ whose rules are "topmost" for a designated kind `[State]` of states. We also assume that an operator $\{\_\}$ is used to enclose states so that all transitions in R have that operator as their top operator in their lefthand sides. What follows can be of course generalized but the former assumptions facilitate the presentation. Finally, we shall assume that $\mathcal{R}$ has good executability properties, i.e., that E is sufficient complete, confluent and terminating modulo A (that is that the equational part correctly defines functions), and R is coherent with E modulo A [5] (that is that applying equations to evaluate functions does not interfere with the application of the rules that dictate the system transitions). Fortunately, the standard Maude tools offer some automatization support for checking such properties.

We consider a well-known semantical domain for rewrite theories, namely, Kripke structures, which are suitable to formulate state space exploration problems like model checking.

Definition 2 (Kripke structure). *A* Kripke structure *K is a tuple $K = (S, \to, L, AP)$ such that S is a denumerable set of states, $\to \subseteq S \times S$ is a transition relation between states, and $L : S \to 2^{AP}$ is a labelling function mapping states into sets of atomic propositions AP (i.e. observations on states).*

The Kripke semantics of a rewrite theory is defined as expected, with `State`-sorted terms as states and one-step rewrites between `State`-sorted terms as transitions. The labelling function is defined by Boolean predicates specified equationally in the rewrite theory. As proved in [7], any computable Kripke structure, even an infinite-state one, can be obtained from an executable rewrite theory using only a finite signature Σ, and finite sets E of equations, A of axioms and R of rules.

Definition 3 (rewrite theory semantics). *Let $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ be a rewrite theory with a designated state sort `State`, symbols and equations for a labelling function $L \in \Sigma$, and atomic propositions AP. The Kripke structure associated to $\mathcal{R}$ is $K_{\mathcal{R}} = (T_{\mathtt{State}/E\cup A}, \to, L_{E/A}, AP)$ such that $T_{\mathtt{State}/E\cup A}$ are all `State`-sorted states, $\to$ is defined as $\{[u] \to [v] \mid \mathcal{R} \vdash u \to^{1}_{R,\mathtt{State}} v\}$ (i.e. transitions are one-step rewrites between $E \cup A$ equivalence classes of `State`-terms in $\mathcal{R}$), and $L_{E/A}$ is the interpretation of L in the initial algebra $\mathcal{T}_{\Sigma/E\cup A}$.*

We will consider bisimulation as the key semantic equivalence.

Definition 4 (bisimulation). *Let $K = (S_K, \to_K, L_K, AP_K)$, $H = (S_H, \to_H, L_H, AP_H)$ be two Kripke structures, and let $\sim \subseteq S_K \times S_H$ be a relation between S_K and S_H. We say that $\sim$ is a* bisimulation *between K and H iff for each two states $s \in S_K$ and $s' \in S_H$ such that $s \sim s'$ we have that: (i) $L_K(s) = L_H(s')$; (ii) $s \to_K r$ implies that there is a state r' s.t. $s' \to_H r'$ and $r \sim r'$; and (iii) $s' \to_H r'$ implies that there is a state r s.t. $s \to_K r$ and $r \sim r'$.*

In what follows we shall focus on bisimulations such that the relation $\sim$ is an equivalence relation. In particular this includes the case of bisimulations induced by symmetries, i.e., when two states are bisimilar if they belong to the same class of symmetric states.

This is precisely the case of those examples we use as a benchmark in the performance evaluation of §5, in which systems are composed by families of replicated concurrent processes with identical behavior. The nature of the symmetry depends very often on the communication topology. For example, in the well-known Peterson's mutual exclusion protocol [10] there is a full symmetry between the processes: we can safely interchange any two processes as long as actual identifiers are not observed. Another example is the well-known Dijktra's Dining Philosophers problem [11], which does not exhibit a full symmetry but a rotational one (due to the circular topology): any rotation of philosophers in a state yields a bisimilar state (if we are not interested in their actual identity).

Equivalence classes of symmetric states can be conveniently defined as the orbits of a group action, which yields the cases of symmetry reductions as special instances of our approach. We hence recall here some basic notions of groups.

Definition 5 (groups, generators and actions). *A* group *is a tuple* $(G, \circ)$ *such that* G *is a set of elements and* $\circ : G \times G \to G$ *is a binary operation on* G *where* $\circ$ *is associative; there is an* identity *element* $e \in G$ *such that* $\forall f \in G. f \circ e = e \circ f = f$*; and each element* $f \in G$ *has an inverse* $f^{-1} \in G$ *such that* $f \circ f^{-1} = f^{-1} \circ f = e$.

Let $(G, \circ)$ *be a group and* $H \subseteq G$ *be a subset of* G*. The group generated by* H *denoted* $\langle H, \circ \rangle$ *is defined as the closure of* H *under the inverse and product operators* $(\_)^{-1}$ *and* $\circ$ *of* G*. In general* $\langle H, \circ \rangle$ *will be a subgroup of* $(G, \circ)$*, but if* $\langle H, \circ \rangle$ *coincides with* $(G, \circ)$*, then* H *is said to* generate G *and its elements are called* generators*.*

Let $(G, \circ)$ *be a group and* A *be a set. A* group action *of* G *on* A *is a monoid homomorphism* $[\![\cdot]\!] : G \to (A \to A)$*, that is,* $[\![f \circ g]\!] = [\![f]\!]; [\![g]\!]$*, where* $f; g$ *denotes function composition in* $(A \to A)$*, and* $[\![e]\!] = id_A$*, with* id_A *the identity on* A*.*

Notable examples are full symmetric and rotation groups, which capture typical symmetries introduced by replication (e.g. of processes with identical behavior) in concurrent systems. Generators define groups in a finite and concise manner. Well known examples of generators for full symmetric and rotation groups are transpositions and single rotations, respectively. Group actions (which allow us to apply group operations to concrete domains like state descriptions) can be defined by means of equations of the form `[[f]](t) = t'` where `f` denotes a group element (possibly a generator) and `t`, `t'` are `State`-sorted terms. Since rewriting logic (and its realization in Maude) allows to handle the axioms of sets (associativity, commutativity, identity), a group action can be described very concisely.

For instance, consider the case of systems defined by multisets of attributed objects and the group of all permutations on object identifiers for which transpositions are generators. The application of transpositions can be defined (by structural induction) with the following equations

```
eq [[i<->j]](none)        = none .
eq [[i<->j]](obj1 conf)   = [[i<->j]](obj1) [[i<->j]](conf) .
eq [[i<->j]](< k | A1 >) = < [[i<->j]](k) | [[i<->j](A1) > .
eq [[i<->j]](i)           = j .
eq [[i<->j]](j)           = i .
eq [[i<->j]](k)           = k [owise] .
```

where `none` denotes the empty set, set union is denoted by juxtaposition and `< k | A1 >` denotes an object with identifier `k` and attributes `A1`. The equations for applying the transpositions to attributes, for example, to object references, are neglected here since they are example-specific.

Such conciseness is crucial for our approach since most of the correctness checks rely on these equations: the fewer equations, the fewer proofs and checks to be carried out.

3 C-Reductions for Kripke Structures

This section introduces the idea of *canonical reductions*, abbreviated c-reductions as a generic means to reduce a Kripke structure K by exploiting some equivalence relation $\sim$ on the states of K which is also a bisimulation on K (i.e. between K and itself). In the next section we will explain how c-reductions are realized and analyzed in rewriting logic, where Kripke structures are specified by rewrite theories.

We start by defining canonizer functions, which are used to compute for a given state a canonical representative of its equivalence class, modulo some equivalence relation which is also a bisimulation (e.g. a canonical permutation of the identifiers of processes with identical behavior).

Definition 6 (canonizer functions). *Let $K = (S, \rightarrow, L, AP)$ be a Kripke structure, and let $\sim \subseteq S \times S$ be a an equivalence relation which is a bisimulation on K. A function $\mathsf{c} : S \rightarrow S$ is a $\sim$-canonizer (resp. strong $\sim$-canonizer) iff for each $s \in S$ we have $s \sim \mathsf{c}(s)$ (resp. $s \sim \mathsf{c}(s)$, and $s \sim s' \rightarrow \mathsf{c}(s) = \mathsf{c}(s')$).*

Canonizer functions are used to compute smaller but semantically equivalent (i.e. bisimilar) Kripke structures by applying canonizers after each transition. Strong canonizers provide unique representatives for the equivalence classes of states and, hence, more drastic space reductions. That is, for two different but equivalent states $s \sim s'$ they provide the same canonical representative (i.e. $\mathsf{c}(s) = \mathsf{c}(s')$). Typical examples of strong canonizers for equivalence classes are functions based on *enumeration* strategies [4] which generate the complete set of states of the equivalence class and then apply some function over it (e.g. based on a total ordering of the states). For example, consider a system where some class of processes exhibit a full symmetry (like the afore-mentioned Peterson's mutual exclusion protocol). Then an enumeration strategy canonizer just generates all states that result from permuting (symmetric) processes in all possible ways and then selects one according to some total order (e.g. the lexicographical order of the description of states). Of course, strong canonizers can be obtained in more efficient and smarter ways. An example is shown in §4.4.

Instead, a non-strong (or weak) canonizer might provide different representatives for equivalent states, that is it might be the case that $\mathsf{c}(s) \neq \mathsf{c}(s')$ even though $s \sim s'$. Weak canonizers provide weaker space reductions but they sometimes enjoy advantages over strong canonizers. For instance, in some cases they are easier to be defined and analyzed, and their computation can be much more efficient in terms of runtime cost. Such heuristic canonizers can be found for instance in [12, 13] where the rough idea is to consider an ordering of the states that depends on part of the state description only. The resulting ordering relation is partial and the representative of a state is computed as one of the least states of the ordering (obtained by iteratively applying some monotonic operation like a transposition).

The reduction of the state space is obtained by applying the canonizer to states after a transition. This is what we call a c-*reduction*.

Definition 7 (c-reduction of a Kripke structure). *Let $K = (S, \rightarrow, L, AP)$ be a Kripke structure, and let $\mathsf{c} : S \rightarrow S$ be a $\sim$-canonizer function for some equivalence relation $\sim \in S \times S$ which is a bisimulation on K. We call the Kripke structure $K_{\mathsf{c}} = (S, (\rightarrow; \mathsf{c}), L, AP)$ the* c-reduction *of K, where the composed transition relation $\rightarrow; \mathsf{c}$ is defined as $\{(s, \mathsf{c}(r)) \in S^2 \mid s \rightarrow r\}$.*

We sometimes decompose transitions $s \rightarrow \mathsf{c}(s')$ in K_{c} to make explicit the canonization step from a state s to its canonical representative $\mathsf{c}(s)$ with $s \rightarrow s' \rightarrow_{\mathsf{c}} \mathsf{c}(s')$.

An important result is then that a c-reduction is bisimulation preserving.

Theorem 1 ($\sim$-preservation). *Let $K = (S, \rightarrow, L, AP)$ be a Kripke structure, let $\sim$ be an equivalence relation on S that is a bisimulation on K, and let c be a $\sim$-canonizer function. Then $\sim$ is a bisimulation relation between K and K_{c}.*

Proof. Let s and s' be two arbitrary states of S such that $s \sim s'$. Condition (i) of Def. 4 holds trivially by the definition of $\sim$.

Condition (ii) is also easy to see. Indeed if we have $s \rightarrow r$, we know that s' can simulate the transition $s \rightarrow r$ by a transition $s' \rightarrow r'$ since $\sim$ is a bisimulation on K. But then we have $r \sim \mathsf{c}(r')$ (since c preserves $\sim$, and $\sim$ is an equivalence relation an hence symmetric and transitive).

$$\begin{tikzcd}
s \arrow[rr] \arrow[d, phantom, "\wr"] & & r \arrow[d, phantom, "\wr"] \arrow[dl, phantom, sloped, "\sim"] \\
s' \arrow[r] & r' \arrow[r, "\mathsf{c}"'] & \mathsf{c}(r')
\end{tikzcd}$$

(i)

$$\begin{tikzcd}
s' \arrow[r] \arrow[d, phantom, "\wr"] & r' \arrow[r, "c"'] \arrow[dr, phantom, sloped, "\sim"] & c(r') \arrow[d, phantom, "\wr"] \\
s \arrow[rr] & & r
\end{tikzcd}$$

(ii)

Condition (iii) is also easy to show (c.f. beside figure (ii)). Indeed, a transition $s' \rightarrow r' \rightarrow_{\mathsf{c}} \mathsf{c}(r)$, can be simulated by some transition $s \rightarrow r$ such that $r \sim r'$ since $\sim$ is a bisimulation on K. But, as in the above case, we also have that $r' \sim \mathsf{c}(r)$ (since c preserves $\sim$ and $\sim$ is an equivalence relation and hence transitive). □

4 C-Reductions in Rewriting Logic

We now describe a methodology for defining and analyzing c-reductions in rewriting logic. We shall often mention how such formalization can benefit from the tool support offered by the Maude framework such as Maude's LTL Model Checker [14], Invariant Analyzer [15], Inductive Theorem Prover [16, 17] and Church Rosser and Coherence Checker [18].

We assume that there is some regularity in $\mathcal{R}$ (e.g. replicated process) that implicitly defines a bisimulation $\sim$ on states (e.g. a permutation of processes) to be exploited to ease the analysis of $\mathcal{R}$. Our methodology is then based on the following steps: (i) implement the regularity correctly; if it is defined by a group action, implement the group action that induces $\sim$ and verify that it is indeed a group action (§4.1) which ensures $\sim$ to be an equivalence relation; (ii) verify that $\sim$ preserves the state predicates (§4.2); (iii) verify that $\sim$ is a bisimulation (§4.3); (iv) define a c-canonizer and show it to be a (possibly strong) $\sim$-canonizer (§4.4); (v) build the c-reduction of $\mathcal{R}$ (§4.5); and (vi) evaluate its performance (§5).

We remark that the steps of this methodology proceed by tackling different levels of abstraction so that they act as building blocks to be re-used whenever needed. For instance, verifying a `c`-reduction strategy does not require performing all the verification steps if it is based on a state equivalence that has been already proven to be correct. In practice, bisimulation relations and their canonizers need not to be defined and proven to be correct for every system, as there will be classes of systems for which they can be specified once and for all. In such cases, one can define reductions as theory transformations for wide classes of examples corresponding, for instance, to certain permutation groups, or to other useful equivalence relations besides the symmetry reduction case. In Maude this can be done by exploiting reflection, so that the `c`-reduction is automatized as a function at the metalevel, possibly after checking some proof obligations. Our current prototype [9], for instance, is able to apply some generic reductions to object based Maude modules.

In addition, in some cases it may be more convenient to bypass some steps, for instance to check the correctness of a canonizer when it is not clear which is the underlying group action (which may not exist at all, since even if we focus here on group-theoretic reductions, `c`-reductions are more general).

A last remark is that even if we put some stress on reductions based on group actions in some of the steps of our methodology, the `c`-reduction technique, in particular steps (v)-(vii), is more general and accepts arbitrary canonizers. The point of focusing on group action is to show an example of generic reduction that is amenable to semi-automatic correctness checks (steps (i)-(iv)).

This paper puts the emphasis on steps (iv) to (vi) which constitute our main contribution here; while steps (i) to (iii) receive less attention since they are work in progress (though useful to illustrate the whole picture of our approach).

4.1 Checking group actions

In the case of equivalences yielded by group actions, one can start defining the group, the generators and their group actions and provide a formal proof of their properties. Many symmetries such as full and rotation symmetries have been identified and thoroughly studied in the past so we won't pay much attention to this aspect in this work. In what follows we will sometimes instantiate our methodology on the particular case of symmetries assuming correctly defined groups, generators and actions.

We assume that the relation on states we have to analyze is equationally defined by topmost equations relating two `State`-sorted terms. In the case of group actions with generators, we can just give equations of the form `[[g]]({t}) = {t'}`, for `g` being a generator (plus the equations defining the inverse operation, unless inverses of generators can be obtained by composition of generators as in the case of transpositions and rotations). Such equations define the actual application of a generator. An example of such equations was provided in §2.

Of course, we need to check that for each generator g and each state u we have $[\![g \circ g^{-1}]\!](u) = u$, but this can be done easily because we can devise inductive proofs exploiting generators. For instance, in the case of full symmetries all we have to do is to prove the equality $[\![i \leftrightarrow j]\!]([\![i \leftrightarrow j]\!](\{t\})) = \{t\}$, i.e. that applying the same transposition of i and j (denoted $i \leftrightarrow j$) twice amounts to applying the identity so that rewrite steps are reversible.

If one has correctly specified a group action, and in particular checked (or given as explicit equations) the monoid homomorphism equations `[[id]]({t}) = {t}` and `[[g∘g']]({t}) = [[g]]([[g']]({t}))` plus the above-mentioned check of the inverses for generators, one has already proved that $\sim$ is an equivalence relation. Indeed a group action implicitly defines an equivalence relation between states as follows $\{t\} \sim \{t'\} \Leftrightarrow \exists f \in G\ s.t. [\![f]\!](\{t\}) = \{t'\}$.

4.2 Checking that $\sim$ preserves atomic predicates

To ensure that $\sim$ preserves the atomic predicates AP under consideration we can proceed as follows.

We first define a rewrite theory $\mathcal{R}_\sim$ for the sole purpose of this analysis defined as $\mathcal{R}_\sim = (\Sigma \cup \Sigma_\sim, E \cup E_\sim \cup A, \{$`{t} => [[g]]({t})`$\mid$ *g is a generator or the inverse of a generator*$\}, \phi)$, where $\Sigma_\sim$ and $E_\sim$ extend the equational part of $\mathcal{R}$ with the definition of the equations for the functions generating the state orbits (e.g. the group generators) for the relation $\sim$. In words, we substitute the rules of $\mathcal{R}$ by rules that allow us to go from a state u to another state v that results from applying a generator (or the inverse of a generator) to u.

If the action group is correctly defined (as we discussed in 4.1, i), it is easy to see (by the properties of the generators of a group) that in this rewrite theory two states are reachable if and only if they are in the same orbit, i.e. that the relation $\sim$ is defined by the rewrite relation $\rightarrow^*_{\mathcal{R}_\sim}$. That is, that $u \sim v$ holds exactly when the above defined rewrite theory can prove $u \rightarrow^*_{\mathcal{R}_\sim} v$.

Moreover, proving that a predicate p is preserved by $\sim$ is exactly proving that p is invariant under $\mathcal{R}_\sim$. Such a proof can be greately facilitated if p is defined by pattern matching, i.e. by topmost unconditional equations of the form `p({t}) = true`.

More formally, if we let $\mathcal{R}_\sim$ be the rewrite theory defined above (and we check that the lefthand sides of all its rules are free constructor terms modulo A), all we have to do is to prove that for each rule `{t'} => [[g]]({t'}` $\in R_\sim$, equation `p({t}) = true` $\in E$, and A-unifier θ between `t'` and `t` (i.e. a mapping θ of variables into non-necessarily ground terms such that $\theta($`t'`$) =_A \theta($`t`$)$), then we have `p({`θ`([[g]]({t})})= true`. If such is the case, indeed, we can conclude that for each pair of states $u, v \in T_{\mathtt{State}_{E/A}}$ it holds that $u \sim v$ implies $\mathtt{p}(u) = \mathtt{p}(v)$, i.e. that `p` is invariant under $\sim$.

Fortunately, Maude has a tool called the Invariant Analyzer [19, 15] (InvA) which can automate a good part of the effort of proving such invariants, leaving the remaining proof obligations for Maude's inductive theorem prover [17].

As a matter of fact, as part of our experimentation we were able to use the tool to check that a simple neighborhood property of a model of the dining philosophers problem ("no two consecutive philosophers eating") is invariant under rotations but not under arbitrary permutations. The first result was obtained in a fully automatic way, while the second follows from an analysis of the few proof obligations provided by InvA.

4.3 Checking that $\sim$ is a bisimulation

Once $\sim$ has been shown to preserve the atomic predicates of interest, all there is left to check is that $\sim$ is a bisimulation.

In our setting this amounts to checking the joinability of suitable "critical pairs" between the state transition rules `{t} => {t'} if cond` in our original set R, and the rules `{t''} => {t'''}` of $\mathcal{R}_{\sim}$. For A-unifiers θ between `t` and `t''` bisimulation is ensured if we prove that for each critical pair denoted with ordinary arrows in the diagram on the right, there is a rule in R giving us a one-step rewrite $\{\theta(\texttt{t'''})\} \to_R \{w\}$ for which we can prove: $\{\theta(\texttt{t'})\} \to^*_{R_\sim} \{w\}$.

$$\begin{tikzcd}
\{\theta(\texttt{t})\} \arrow[r, "R"'] \arrow[d, "R_\sim"'] & \{\theta(\texttt{t'})\} \arrow[d, dotted, "R_\sim"', "*"] \\
\{\theta(\texttt{t'''})\} \arrow[r, dotted, "R"'] & \{w\}
\end{tikzcd}$$

Part of this could be automatized by using Maude's automated reachability analysis capabilities, possibly including narrowing-based ones since some of the above terms might not be ground (i.e. they might contain variables). Of course, it is not as easy as it sounds, since the above proofs are inductive, so that sometimes it may not be straightforward to show these properties for all ground terms just by rewriting terms with variables: some inductive arguments may also be needed.

4.4 Defining and analyzing canonizer functions

The next step is to define canonizer functions `c : [State]` $\to$ `[State]` in a protecting extension of the rewrite theory $\mathcal{R}$ under study. Note that in order to define the function `c : [State]` $\to$ `[State]` we may need to define some auxiliary functions (for example, the ordering relations used in symmetry reductions to determine a representative for each orbit).

Definition 8 (c-extension of a rewrite theory). *Let $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ be a rewrite theory and `c` a canonizer function. The `c`-extension of $\mathcal{R}$ is the rewrite theory $\mathcal{R}^{\mathtt{c}} = (\Sigma \uplus \Sigma_{\mathtt{c}}, E \cup G_{\mathtt{c}} \cup A, R, \phi_c)$ where $\mathtt{c} \in \Sigma_{\mathtt{c}}$, and $\Sigma_{\mathtt{c}} \setminus \{\mathtt{c}\}$ are other auxiliary functions; $(\Sigma \uplus \Sigma_{\mathtt{c}}, E \cup G_{\mathtt{c}} \cup A)$ is a protecting extension of $(\Sigma, E \cup A)$, and is ground confluent, ground terminating, and sufficiently complete w.r.t. the same signature of constructors; and $\phi_{\mathtt{c}}$ extends ϕ with frozenness maps for all arguments of all functions in $\Sigma_{\mathtt{c}}$.*

Imposing frozenness on the operators of $\Sigma_{\mathtt{c}}$ is needed for the coherence result of Lemma 1.

For a given bisimulation $\sim$ many candidate canonizers may exist, each leading to different results in terms of the size of the reduced state space and computational performance.

In any case, all canonizer functions must enjoy correctness properties such as the extended equational theory being confluent, terminating, sufficiently complete with respect to constructors (which may be the same constructors as those of

$(\Sigma, E \cup A)$). More importantly (for what regards our contribution), canonizer functions must preserve $\sim$, i.e. they must be $\sim$-canonizers. This might require some theorem proving but it can be relatively easy to check in most cases, since we can use the equations G_c and show that each one preserves $\sim$.

For example, in the case of *local* reduction strategies [4] the equations $G_{\mathtt{c}}$ defining `c` are of the form `c({t}) = c([[g]]({t})) if [[g]]({t})<{t}` for each possible generator (or inverse generator) `g` with `_<_` defining an ordering relation on states, plus an equation `c({t})={t} [owise]` to deal with the case when none of the previous equations is applicable (denoted by the keyword `[owise]`), that is when there is no way to transform a state into a *smaller* equivalent one by applying a generator. Since such equations define `c` in terms of group actions or of the identity function when all conditions fail, preservation of $\sim$ is immediate by the very definition of `c`.

Examples of such local search strategies are implemented in our prototype tool [9]. For example the following lines (adapted from [9] for the sake of illustration) report the main equations defining a weak `c`-reducer that keeps permuting the identity of two objects as long the resulting state is lower according to some total ordering relation defined by `<` [3] .

```
ceq c( < id1 | A1 > < id2 | A2 > conf1 )
    = c([[id1<->id2]]( < id1 | A1 > < id2 | A2 > conf1 )
    if id1 < id2  /\ [[id1<->id2]](A2) < A1 .
eq c(conf1) = conf1 [ owise ] .
```

Note that if `<` is a total order, then the equations are confluent and the canonizer is strong.

A very similar situation is that of *enumeration* strategies [4] where canonizers are defined as `c({t}) = min{[[f]]({t})}` where `f` ranges over all elements of the group under consideration. Again, preservation of $\sim$ by c follows from the very definition of c. Indeed, for all states u, $\mathtt{c}(u)$ will be necessarily of the form $[\![g_1 \circ g_2 \circ \cdots \circ g_n]\!](u)$, with each g_i being a generator. We call the equation format described in this paragraph *group application form.* All the `c`-reduction strategies evaluated in §5 are essentially of these forms.

Proposition 1. *Let $\mathcal{R}$ be a rewrite theory, $\mathcal{R}_\sim$ be defined as usual, and $\mathcal{R}^{\mathtt{c}}$ be the `c`-extension of $\mathcal{R}$ such that the equations of $E_{\mathtt{c}}$ defining `c` are in* group application form*. Then, for all states $u \in T_{\mathtt{State}/E\cup A}$ we have that $\mathtt{c}(u) \sim u$.*

In practice, when implementing `c`-strategies in the above form, all we have to check are the ordinary well-definedness properties of the equations of `c`: termination and sufficient completeness with respect to constructors, for which one can rely on the standard Maude tools.

[3] Such order takes into account the ordering of the identities `id1`, `id2` first and the order of their attributes second. In our tool `<` is based on the lexicographical order of the meta-representation of terms by default but it can be customized at the object level to achieve better results in terms of performance as we show in §5.

Note that proving ground confluence of `c` is not sufficient to show that `c` is a strong canonizer. It may still be the case that for some two states u, v such that $u \sim v$ we have that $\mathtt{c}(u) \neq \mathtt{c}(v)$. Indeed, what we need to prove to show that `c` is a strong canonizer is that for all generators g and states s we have $\mathtt{c}(s) = \mathtt{c}([\![g]\!](s))$. It is easy to see that if the previous property holds, an inductive argument allows us to conclude $\mathtt{c}(s) = \mathtt{c}([\![f]\!](s))$ for any possible group operation f and hence for any two equivalent states $s \sim s' = [\![f]\!](s)$.

But the above check can be reduced to an ordinary (automatic) confluence check if we extend $E_{\mathtt{c}}$ with the set of equations `test(g,c({t})) = c([[g]]({t}))` and `test(g,c({t})) = c(({t}))`, for all generators `g` (and inverse generators) of the group under consideration, where `test` is an auxiliary function symbol that ensures that the generator is applied only once. We denote this extension of $E_{\mathtt{c}}$ by $E_{\mathtt{c}^{\sim}}$. In words, the idea is that applying or not a group generator and then canonizing should result in the very same state.

Proposition 2. *Let $\mathcal{R}$ be a rewrite theory, $\mathcal{R}_{\sim}$ be defined as usual, $\mathcal{R}^{\mathtt{c}}$ be the* c*-extension of $\mathcal{R}$, and let $E_{\mathtt{c}^{\sim}}$ be defined as above. If $(\Sigma \cup \Sigma_{\mathtt{c}}, E \cup E_{\mathtt{c}^{\sim}}, A)$ is confluent, then* `c` *is a strong canonizer.*

This result paves the way for the use of Maude's confluence checker [18]. Of course, there are cases in which no check is needed. For instance, it is well-known that enumeration strategies yield strong canonizers, while local strategies are not strong in general.

4.5 Building c-reductions

The next step is to build the c-reduction. Recall that we have defined the c-reduction $\mathcal{K}_{\mathtt{c}}$ of a Kripke structure $\mathcal{K}$ in Def. 7. Starting from a system specification given as a rewrite theory $\mathcal{R}$, our goal is to build $\mathcal{K}_{\mathtt{c}}(\mathcal{R})$, but this can be easily done by applying a theory transformation mapping $\mathcal{R}$ into its c-reduction $\mathcal{R}_{\mathtt{c}}$, which is defined as follows.

Definition 9 (c-reduction of a rewrite theory). *Let $\mathcal{R}^{\mathtt{c}} = (\Sigma \cup \Sigma_{\mathtt{c}}, E \cup G_{\mathtt{c}} \cup A, R, \phi_{\mathtt{c}})$ be the* c*-extension of a rewrite theory $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$. We call the rewrite theory $\mathcal{R}_{\mathtt{c}} = (\Sigma \cup \Sigma_{\mathtt{c}}, E \cup G_{\mathtt{c}} \cup A, R_{\mathtt{c}}, \phi_{\mathtt{c}})$ where $R_{\mathtt{c}} = \{$`t => c(t') if cond` $\mid$ (`t => t' if cond`) $\in R\}$ a* c-reduction *of $\mathcal{R}$.*

In words, $\mathcal{R}_{\mathtt{c}}$ is very much like $\mathcal{R}$, but each rule `t => t' if cond` in R, is transformed into a rule `t => c(t') if cond`, i.e., into a rule where the canonizer function `c` is applied to the right hand side to ensure that canonization is performed after each system transition.

This step is supported by our prototype [9], though limited to a particular class of rewrite theories (those using the object based signature). The implementation exploits Maude reflective features: the transformation is defined by a function that manipulates the meta-representation of the theory to be c-reduced.

Note that in some cases it might be convenient not to apply the canonizer after each step. For instance if we know that the rule will always result in a canonical state we can save the time of applying the function.

It is trivial to show that $\mathcal{R}_{\mathsf{c}}$ is a c-reduction by construction, and in particular that $\mathcal{K}_{\mathsf{c}}(\mathcal{R}) = \mathcal{K}(\mathcal{R}_{\mathsf{c}})$. It can also be shown that it has good executability properties. By the properties required for G_{c}, it inherits all the properties of the equational part of $\mathcal{R}$, namely sufficient completeness, confluence and termination modulo A. What is left to show is that $\mathcal{R}_{\mathsf{c}}$ is coherent modulo A.

Lemma 1 (coherence of $\mathcal{R}_{\mathsf{c}}$). *Let $\mathcal{R} = (\Sigma, E \cup A, R, \phi)$ be ground confluent, terminating, sufficiently complete, and with R ground coherent with E modulo A. Then $\mathcal{R}_{\mathsf{c}}$ defined as above has R_{c} ground coherent with $E \cup G_{\mathsf{c}}$ modulo A.*

Proof. Since $\mathcal{R}$ is topmost and all operators in Σ_{c} are frozen (see Def. 8), any ground $\Sigma \cup \Sigma_{\mathsf{c}}$-term u of sort `State` such that $\mathcal{R} \vdash u \to^1 v$ must be a Σ term. Therefore, its $E \cup G_{\mathsf{c}}/A$-canonical form u' is exactly its E/A-canonical form. By the ground coherence of R with E modulo A we obtain the inner pentagon in the diagram on the right which trivially yields the outer pentagon, proving $E \cup G_{\mathsf{c}} \vdash \mathsf{c}(v) = \mathsf{c}(v')$ as desired. □

$$\begin{tikzcd}
u \arrow[r, "1", "\mathcal{R}"'] \arrow[dd, "E/A!"'] & v \arrow[rr, "\mathsf{c}"'] \arrow[dr, "E/A"', "*"] & & \mathsf{c}(v) \arrow[dr, "E/A"', "*"] & \\
& & w & & \mathsf{c}(w) \\
u' \arrow[r, "1", "\mathcal{R}"'] & v' \arrow[rr, "\mathsf{c}"'] \arrow[ur, "E/A", "*"'] & & \mathsf{c}(v') \arrow[ur, "E/A"', "*"] &
\end{tikzcd}$$

Theorem 2 (executability of $\mathcal{R}_{\mathsf{c}}$). *Let $\mathcal{R} = (\Sigma \cup c, E \cup G_c \cup A, R, \phi)$ be a rewrite theory with good executability properties, then $\mathcal{R}_{\mathsf{c}}$ has good executability properties.*

Proof. The proof immediatedly follows from the fact that $\mathcal{R}_{\mathsf{c}}$ has the same equational part as $\mathcal{R}_{\mathsf{c}}$ and from Lemma 1. □

5 Performance experiments

We present here a subset of the experiments we carried out, with the main purpose of validating the effectiveness of the implementation of the c-reduction approach in Maude.

Our main hypothesis to be checked is that the relative performance gain (in terms of runtime and state space reductions) is comparable to the one obtained by state-of-the-art model checkers. The second hypothesis is that c-reductions are more efficient than the previous approach to symmetry reductions in Maude described in [8]. Finally, we enrich the experiments where we combine various c-reductions not supported by the previously mentioned tools. The Maude implementation of the benchmark examples used here are included in the release of our prototype [9].

Comparison with SymmSPIN We have chosen SymmSPIN [12, 13] as a representative model checker with which to compare our approach. SymmSPIN extends the SPIN [3] model checker with support for symmetry reductions. It is worth to remark that we do not perform absolute comparison as we aim at

			SymmSpin				c-reductions			
			weak		strong		weak		strong	
Experiment	n	if	sf	tf	sf	tf	sf	tf	sf	tf
Peterson	2	-0.50	-0.48	+0.00	-0.49	+0.00	-0.45	+0.00	-0.45	+0.00
	3	-0.83	-0.75	-0.79	-0.82	-0.79	-0.77	+0.00	-0.83	+1.50
	4	-0.96	-0.91	-0.77	-0.95	-0.84	-0.93	-0.75	-0.95	+0.50
DBM	7	-0.99	-0.99	-0.99	-0.99	-0.80	-0.98	-0.87	-0.99	+2.00
	8	-0.99	-0.99	-0.99	-0.99	-0.48	-0.99	-0.95	-0.99	-0.65

Table 1. SymmSpin vs c-reductions in Maude.

checking the usefulness of our approach (experiments with SymmSPIN cannot be reproduced since the tool is no more available for download).

We have implemented in Maude two of the benchmark models tested in [12, 13], namely Peterson's mutual exclusion protocol [10], and a database management system [20]. Both examples exhibit a full symmetry due to the presence of families of replicated concurrent processes with identical behavior. In both cases we manually translated the Promela specification into Maude specifications, in the most faithful manner we could.

We have considered various c-reduction strategies. For the sake of simplicity, we consider only the two best strategies of SymmSPIN (i.e. *pc-segmented* and *pc-sorted*) and the two best of our own strategies our implementation (partly inspired by the ones proposed in [12, 13]), for which regards time and state space reduction. We call them *strong* and *weak* as they are actually strong and non-strong canonizers.

Table 1 presents the results for instances of the models with increasing number of components (n). We offer only results for those instances for which it has been possible to generate the unreduced state spaces, so to compare the relative gain of the reductions in terms of relative state reduction factor (sf) and relative time reduction factor (tf). A relative state (resp. time) reduction factor of k indicates that, if the non-reduced exploration involved n states (resp. seconds) then the reduced exploration involved $m = n + (k \times n)$ states (resp. seconds). Put it otherwise, k is calculated as $k = (m/n) - 1$. Clearly, values below 0 indicate a reduction, while numbers above 0 an increase. The table also includes the "ideal" relative reduction factor (if), which in the case of full symmetries is $(1/n!) - 1$ where n is the size of the permutations (i.e. the number of replicated processes in the examples), since the size of each orbit is at most $n!$. So for an exhaustive exploration that requires s states and t seconds an ideal factor k means that we can expect the reduction to require at the best the exploration of $n + (k \times s)$ states in $t + (k \times t)$ seconds (where k is obviously negative). Clearly, "sf" and "tf" are always greater than "if". We highlight cells corresponding to the best results in each category (state space and run-time gain) for each model instance. It is worth to remark that the results for the SymmSPIN tool have been derived from [12]. Reproducing them was not possible since SymmSPIN is not available

				Reflection-based			c-reductions					
Params		Not reduced					weak			strong		
n	m	states	if	states	sf	tf	states	sf	tf	states	sf	tf
2	5	38,029	-0.50	19,295	-0.49	+9.62	21,630	-0.43	+1.14	19,025	-0.50	+0.46
3	2	72,063	-0.83	13,280	-0.82	+7.30	29,534	-0.59	+1.01	12,235	-0.83	+0.34
3	3	952,747	-0.83	174,428	-0.81	+5.65	307,532	-0.68	-0.50	160,121	-0.83	+0.30

Table 2. Reflection-based symmetry reduction vs. `c`-reductions in Maude.

for download. Fortunately, we are not interested in absolute measures but in the relative gain of the reduction.

The two approaches provide state space reductions near to the ideal gain. The two strategies based on `weak` canonizers provide very similar outcomes, while the ones based on `strong` canonizers reduce similarly. SymmSPIN is more time-efficient, which is not a surprise, since the reduction algorithms are implemented in a procedural language (C) and efficiently compiled, while our implementation is based on a declarative language (Maude) running over an intepreter (the Maude engine).

Comparison with reflection-based symmetry reduction in Maude Our second set of experiments aims at checking whether `c`-reductions offers better performances than the symmetry reduction implementation in Maude described in [8]. Very briefly, the main idea of [8] is to select the canonical representative of a state on the basis of the lexicographical order of the meta-representation of the state, which is achieved by exploiting Maude's reflection capabilities.

The comparison is performed over the Chain-Replication protocol used in [8]. As in the previous case, the replication of identical processes yields a full symmetry. Table 2 presents our results in the same format as Table 1 with the only exception that the model is instantiated with two parameters: the number n of replicated components and the number m of queries they perform. The table shows that the reductions of the reflection-based approach of [8] are bounded by the ones obtained by our strategies. That is: our weak strategy (resp. strong) provides a better (resp. worse) time reduction factor, while our strong (resp. weak) strategy provides a better (resp. worse) space reduction factor. In particular our weak strategy offers worse space reductions, while the strong one offers better space reductions. More interestingly, our reduction strategies introduce much less time overhead, differing often by an order of magnitude. This is not a surprise, since resorting to Maude's meta-level involves a considerable overhead.

Canonizers based on the lexicographical order of meta-representations are also implemented in our prototype [9] where similar results can be observed.

Exploiting permutations, rotations, reuse and abstraction Our last set of experiments regards the joint application of a number of `c`-reductions of different nature, not supported by the previously mentioned tools. As a test case we have considered a message-passing solution to the Dining Philosophers problem

	NR		NR+RS+FS		NA		NA+RS	
n	states	time	states	time	states	time	states	time
2	21	0	10	0	18	0	10	0
3	115	8	27	12	76	0	27	8
4	801	100	86	60	322	20	86	48
5	6,251	1,456	275	320	1,364	124	275	248
6	54,869	20,765	982	1,732	5,778	740	982	1,412
7	541,731	463,080	3,499	11,828	24,476	6,624	3499	8,124
8	O.T.	O.T.	13,016	49,651	103,682	29,329	13,006	35,594
9	O.T.	O.T.	48,828	247,987	439,204	192,072	48,819	186,329

Table 3. c-reductions for the dining philosophers.

along the lines of the case study used in [21], where newly generated messages (representing forks) receive fresh identities (as the original purpose of [21] was to reason about individual messages). There are a couple of regularities that can be exploited in the form of c-reductions and that happen to yield bisimulations (for an empty set of atomic predicates): the rotational symmetry of philosophers, the full symmetry of messages, the reuse of message identifiers and their abstraction. Of course, the situation is different when one considers state predicates that involve the identity of philosophers or messages. However, our goal here is to validate the effectiveness of the mentioned c-reductions.

Table 3 reports the results. The table presents the size of the state space and the time (in ms) to generate it, for instances of the model with increasing number of philosophers. The table considers the state spaces generated in the following cases: reuse of message identifiers (NR), reuse together with (rotational and full) symmetry reduction (NR+RS+FS), abstraction of name identifiers (NA), and abstraction of name identifiers together with (rotational) symmetry reduction (NA+RS). The sizes of the unreduced state spaces are not shown since they are infinite (due to the creation of messages with fresh identifiers).

The first clear advantage is that reuse of message identifiers yields finite state spaces (since the number of messages in each state is bounded by n). Besides this, we see how combining various c-reductions results in better and more efficient reductions. To be noticed is the fact that name reusing alone ran out of time (more than 5 hours) for models instantiated with more than 7 philosophers, while combining it with symmetry reductions (for messages and philosophers) allows us to manage larger instances. In particular, the best reductions are obtained with the combination of name abstraction and rotational symmetry (NA+RS), while name abstraction alone (NA) offers the fastest explorations from 2 to 8 philosophers, and is outperformed by the combination NA+RS for greater instances.

6 Conclusion

We have presented *c-reductions*, a state space reduction technique, whose main idea is to use canonizer functions mapping each state into a canonical representative of its equivalence class modulo a bisimulation equivalence relation, capturing some specific system regularities. The main differentiating features with respect to other state space reduction techniques are (i) reductions are defined using the system description language; (ii) (re)use of the standard techniques of the verification setting to check correctness of the reduction; (iii) automatization: both for applying the reduction and for checking its correctness; and (iv) generality: it subsumes in a uniform way symmetry reduction as well as other kinds of reductions (name reuse, abstraction).

We have presented the basic infrastructure of our approach, described some typical classes of reductions, illustrated how they can be analyzed and presented some performance results.

Our methodology consists on performing a series of incremental verification steps §4.1-§4.5 which include checking that the equivalence relation is a bisimulation and that reduction strategies preserve such equivalence relation. In this paper we have focused on steps §4.4-§4.5. Steps §4.1-§4.4 are subject of current work but we have offered a preliminary sketch of our ideas.

It is worth to remark that these steps are not to be repeated every time a system is going go be verified. The idea is rather to consider those steps as incrementally providing results to be reused again and again.

We also presented a representative subset of the performance experiments we have carried out. In all cases, regardless of the chosen c-reduction strategy, the c-reductions allow us to explore larger state-spaces. In particular, considering full symmetries, strategies based on the strong canonizers offer in general greater reductions, at the cost of more time, which in some cases is compensated by the smaller sizes of the obtained state spaces. Strategies based on weak canonizers require less application-specific code and less comparisons (e.g. since they are based on an ordering relation on states that takes into account part of the state descriptions only), but offer weaker reductions.

We have observed a comparable performance with respect to mature tools such as SPIN and performance gains with respect to a previous implementations of symmetry reductions in Maude [8]. Moreover, the flexibility of our approach has allowed us to define a wide range of reductions. Beyond the classical permutation and rotation symmetries, we have considered some simple cases of name reuse and name abstraction, which are crucial to deal with the infinite state spaces of systems with dynamic allocation of resources. Compared to the approach presented in [12, 13] we are able to treat a wider class of systems, where identifiers of symmetric objects can appear as pointers in attributes of other objects, and with wider classes of symmetries such as rotational ones. Similar remarks can be made about the approach presented in [8], with respect to which we offer a wider class of reduction strategies and better performance.

A closely related work is the automatic symmetry detection of [4] which also provides an alternative symmetry reduction extension (TopSPIN) of the

SPIN model checker. Our approach does not consider automatic detection of symmetries but, instead, user-definable ones, together with a methodology to check their correctness, with the main advantage being that we rely on tools and techniques used to perform the verification of the system itself. Both lines of research are essentially complementary and can converge in a synergistic way.

We are also investigating how to apply our approach to improve the efficiency of rewriting-logic based interpreters of programming languages with primitives for dynamic memory allocation, and to further explore the combination of c-reductions with other state space reduction techniques, with a particular attention to those already proposed in the setting of rewriting logic and Maude, such as partial order reduction [22], and abstraction [23].

The closest of such approaches is probably [23] where abstractions are defined equationally. The main difference with our approach is in the kind of behavioral equivalence considered: equational abstractions yield simulations while our approach focuses on bisimulations. Another significant difference is that in our approach one has some control on when to apply a reduction, i.e. the c-reduction procedure might be applied to some of the rules only, excluding for instance those rules that are known to yield representative states (e.g. if they affect a part of the state that does not play a role in a symmetry). In that way some unnecessary computations can be spared. A more detailed comparison is under study.

Current efforts are also devoted to a deeper investigation of state space reduction techniques based on name-reuse. Such techniques have a limited impact on Kripke models, where object identifiers are "global", in the sense that they represent an entity through all the states of a model. These techniques become instead fundamental using more sophisticated models (together with non-propositional property specification languages), where object identifiers are local to single states, and "trans-states identities" are explicitly represented through suitable mappings associated to state transitions. Notable examples are History Dependent Automata [24], High-level Allocational Büchi Automata [21], Graph Transition Systems [25] and Counterpart models [26].

We are also working further (and experimenting) on the verification steps that we could only sketch in this paper (steps §4.1-§4.3). Even if we put some stress on reductions based on group actions the c-reduction our approach is more general and accepts any possible canonizer function. Appropriate correctness check methodologies for such cases should however be developed. Particularly interesting could be the definition of c-reductions inspired by process algebra techniques for the minimization of processes modulo bisimulation and fort the axiomatisation of bisimulation relations.

Another possible extension the treatment of equivalence preorder like simulations (as we mentioned when discussing the relation with [23]), and further develop the automatization of our approach and the use of the integrated Maude formal environment [27].

A preliminary version of our tool is publicly available [9]. At the current stage the tools is limited to the application of a set of generic c-reductions to object-based Maude modules.

Acknowledgements. We are grateful to the reviewers for their helpful criticism and detailed comments, and to Dragan Boŝnaĉki for a detailed discussion of our work and its comparison with symmetry reduction techniques.

References

1. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999)
2. Wahl, T., Donaldson, A.F.: Replication and abstraction: Symmetry in automated formal verification. Symmetry **2** (2010) 799–847
3. Holzmann, G.: The SPIN model checker: primer and reference manual. Addison-Wesley Professional (2003)
4. Donaldson, A.F., Miller, A.: A computational group theoretic symmetry reduction package for the SPIN model checker. In: Algebraic Methodology and Software Technology. (2006) 374–380
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude. Volume 4350 of LNCS. Springer (2007)
6. Meseguer, J.: Conditional rewriting logic as a united model of concurrency. Theoretical Computer Science (TCS) **96** (1992) 73–155
7. Meseguer, J., Palomino, M., Martí-Oliet, N.: Algebraic simulations. Journal of Logic and Algebraic Programming **79** (2010) 103–143
8. Rodríguez, D.E.: Combining techniques to reduce state space and prove strong properties. In: Proceedings of the 7th International Workshop on Rewriting Logic and its Applications (WRLA 2008). Volume 238(3) of ENTCS. (2009) 267 – 280
9. C-Reducer, `http://sysma.lab.imtlucca.it/tools/c-reducer`.
10. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc. (1996)
11. Dijkstra, E.W.: Hierarchical ordering of sequential processes. Acta Informaticae **1** (1971) 115–138
12. Bosnacki, D., Dams, D., Holenderski, L.: A heuristic for symmetry reductions with scalarsets. In: International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity (FME'01), Springer (2001)
13. Bosnacki, D., Dams, D., Holenderski, L.: Symmetric SPIN. International Journal on Software Tools for Technology Transfer (STTT) **4** (2002) 92–106
14. Eker, S., Meseguer, J., Sridharanarayanan, A.: The maude ltl model checker and its implementation. In Ball, T., Rajamani, S.K., eds.: SPIN. Volume 2648 of Lecture Notes in Computer Science., Springer (2003) 230–234
15. The Maude Invariant Analyzer Tool (InvA), `camilorocha.info/software/inva`.
16. Clavel, M., Palomino, M., Riesco, A.: Introducing the ITP tool: a tutorial. Journal of Universal Computer Science **12** (2006) 1618–1650
17. Maude Interactive Theorem Prover, `maude.cs.uiuc.edu/tools/itp/`.
18. Durán, F., Meseguer, J.: A Church-Rosser Checker Tool for Conditional Order-Sorted Equational Maude Specifications. In Ölveczky, P.C., ed.: Proceedings of the 8th International Workshop on Rewriting Logic and its Applications (WRLA'10). Volume 6381 of LNCS., Springer (2010) 69–85
19. Rocha, C., Meseguer, J.: Proving safety properties of rewrite theories. In Corradini, A., Klin, B., Crstea, C., eds.: Algebra and Coalgebra in Computer Science. Volume 6859 of LNCS. Springer (2011) 314–328
20. Valmari, A.: Stubborn sets for reduced state generation. In: Proceedings on Advances in Petri nets 1990, Springer (1991)

21. Distefano, D., Rensink, A., Katoen, J.P.: Model checking birth and death. In Baeza-Yates, R.A., Montanari, U., Santoro, N., eds.: 2nd International Conference on Theoretical Computer Science (TCS'02). Volume 223., Kluwer (2002)
22. Farzan, A., Meseguer, J.: Partial order reduction for rewriting semantics of programming languages. In: Proceedings of the 6th International Workshop on Rewriting Logic and its Applications (WRLA'06). Volume 176 of ENTCS. (2007) 61–78
23. Meseguer, J., Palomino, M., Martí-Oliet, N.: Equational abstractions. Theoretical Computer Science **403** (2008) 239–264
24. Montanari, U., Pistore, M.: Structured coalgebras and minimal HD-automata for the π-calculus. Theoretical Computer Science **340** (2005) 539–576
25. Rensink, A.: Isomorphism checking in GROOVE. ECEASST **1** (2006)
26. Gadducci, F., Lluch Lafuente, A., Vandin, A.: Counterpart semantics for a second-order μ-calculus. In Ehrig, H., Rensink, A., Rozenberg, G., Schürr, A., eds.: 5th International Conference on Graph Transformation (ICGT'10). Volume 6372 of LNCS. Springer (2010) 282–297
27. The Maude Formal Environment, `http://maude.lcc.uma.es/MFE/`.