Adding Symmetry Reduction to UPPAAL*

Martijn Hendriks[1] Gerd Behrmann[2] Kim Larsen[2]
Peter Niebert[3] Frits Vaandrager[1]

[1]*Nijmegen Institute for Computing and Information Sciences,*
University of Nijmegen, The Netherlands
{martijnh,fvaan}@cs.kun.nl

[2]*Department of Computing Science,*
Aalborg University, Denmark
{behrmann,kgl}@cs.auc.dk

[3]*Laboratoire d'Informatique Fondementale, CMI,*
Université de Provence, France
peter.niebert@lif.univ-mrs.fr

Abstract

We describe a prototype extension of the UPPAAL real-time model checking tool with symmetry reduction. The symmetric data type *scalarset*, which is also used in the MURφ model checker, was added to UPPAAL's system description language to support the easy static detection of symmetries. Our prototype tool uses *state swaps*, described and proven sound earlier by Hendriks, to reduce the space and memory consumption of UPPAAL. Moreover, under certain assumptions the reduction strategy is *canonical*, which means that the symmetries are optimally used. For all examples that we experimented with (both academic toy examples and industrial cases), we obtained a drastic reduction of both computation time and memory usage, exponential in the size of the scalar sets used.
Keywords: real-time, timed automata, symmetry, model checking.
AMS subject classification: 68Q60.
CR subject classification: D.2.4, F.3.1.

1 Introduction

Model checking is a semi-automated technique for the validation and verification of all kinds of systems [9]. The approach requires the construction of a *model* of the system and the definition of a *specification* for the system. A model checking tool then computes whether the model satisfies its specification. Nowadays, model checkers are available for many application areas, e.g., hardware systems [12, 24], finite-state distributed systems [19], and timed and hybrid systems [23, 30, 27, 18].

Despite the fact that model checkers are relatively easy to use compared to manual verification techniques or theorem provers, they are not being applied on a large

*This work has been supported by the European Community Project IST-2001-35304 (AMETIST), *http://ametist.cs.utwente.nl/*. An extended abstract of this paper appeared as [17].

scale. An important reason for this is that they must cope with the *state space explosion* problem, which is the problem of the exponential growth of the state space as models become larger. This growth often renders the mechanical verification of realistic systems practically impossible: there just is not enough time or memory available. As a consequence, much research has been directed at finding techniques to fight the state space explosion. One such a technique is the exploitation of behavioral symmetries [20, 25, 22, 21, 14, 10]. The exploitation of *full* symmetries can be particularly profitable, since its gain can approach a factorial magnitude.

There are many timed systems which clearly exhibit full symmetry, e.g., Fischer's mutual exclusion protocol [1], the CSMA/CD protocol [26, 30], industrial audio/video protocols [15], and distributed algorithms, for instance [5]. Motivated by these examples, the work presented in [16] describes how UPPAAL, a model checker for networks of timed automata [23, 4, 3], can be enhanced with symmetry reduction. The present paper puts this work to practice: a prototype of UPPAAL with symmetry reduction has been implemented. The symmetric data type *scalarset*, which was introduced in the MURφ model checker [12], was added to UPPAAL's system description language to support the easy static detection of symmetries. Furthermore, the *state swaps* described and proven sound in [16] are used to reduce the space and time consumption of the model checking algorithm. The reduction strategy is optimal under certain assumptions that essentially concern the discrete part of the state only. Thus, the dense time domain does not add extra complexity to the symmetry reduction technique. Run-time data is reported for the examples mentioned above, showing that symmetry reduction in a timed setting can be very effective.

Related work. Symmetry reduction is a well-known technique to reduce the resource requirements for model checking algorithms, and it has been successfully implemented in model checkers such as MURφ [12, 21], SMV [24], and SPIN [19, 8]. As far as we know, the only model checker for timed systems that exploits symmetry is RED [27, 28]. The symmetry reduction technique used in RED, however, gives an over approximation of the reachable state space (this is called the *anomaly of image false reachability* by the authors). Therefore, RED can only be used to ensure that a state is *not* reachable when it is run with symmetry reduction, whereas symmetry enhanced UPPAAL can be used to ensure that a state is reachable, or that it is not reachable.

Contribution. We have added symmetry reduction as used within MURφ, a well-established technique to combat the state space explosion problem, to the real-time model checking tool UPPAAL. For researchers familiar with model checking it will come as no surprise that this combination can be made and indeed leads to a significant gain in performance. Still, the effort required to actually add symmetry reduction to UPPAAL turned out to be substantial. The soundness of the symmetry reduction technique that we previously developed for UPPAAL does not follow trivially from the work of Ip and Dill [21] since the description languages of UPPAAL and MURφ, from which symmetries are extracted automatically, are quite different. In fact, the proof that symmetry reduction for UPPAAL is sound takes up more than 20 pages in [16]. The main technical contribution of the present work is an efficient algorithm for the computation of a representative that – under certain assumptions – is optimal. This is not trivial due to UPPAAL's symbolic representation of sets of clock valuations. Many timed systems exhibit symmetries that can be exploited by our methods. For all examples that we experimented with, we

obtained a drastic reduction of both computation time and memory usage, exponential in the size of the scalar sets used.

Outline. Section 2 presents a very brief summary of model checking and symmetry reduction in general, while Sections 3 and 4 introduce symmetry reduction for the UPPAAL model checker in particular. In Section 5, we present run-time data of UPPAAL's performance with and without symmetry reduction, and Section 6 summarizes and draws conclusions.

2 Model Checking and Symmetry Reduction

This section briefly summarizes the theory of symmetry presented in [21], which we reuse in a timed setting since (i) it has proven to be quite successful, and (ii) it is designed for reachability analysis, which is the main purpose of the UPPAAL model checker. We simplify (and in fact generalize) the presentation of [21] using the concept of bisimulations.

In general, a transition system is a tuple (Q, Q_0, Δ), where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, and $\Delta \in Q \times Q$ is a transition relation between states. Figure 1 depicts a general forward reachability algorithm which, under the assumption that Q is finite, computes whether there exists a reachable state q that satisfies some given property ϕ (denoted by $q \models \phi$).

```
(1)   passed := ∅
(2)   waiting := Q0
(3)   while waiting ≠ ∅ do
(4)         get q from waiting
(5)         if q ⊨ φ then return YES
(6)         else if q ∉ passed then
(7)               add q to passed
(8)               waiting := waiting ∪ { q' ∈ Q | (q, q') ∈ Δ }
(9)         fi
(10)  od
(11)  return NO
```

Figure 1: A general forward reachability analysis algorithm.

Due to the state space explosion problem, the number of states of a transition system frequently gets too big for the above algorithm to be practical. We would like to exploit structural properties of transition systems (in particular symmetries) to improve its performance. Here the well-known notion of bisimulation comes in naturally:

Definition 2.1 (Bisimulation) *A* bisimulation *on a transition system (Q, Q_0, Δ) is a relation $R \subseteq Q \times Q$ such that for all $(q, q') \in R$,*

1. *$q \in Q_0$ if and only if $q' \in Q_0$,*
2. *if $(q, r) \in \Delta$ then there is an r' such that $(q', r') \in \Delta$ and $(r, r') \in R$,*
3. *if $(q', r') \in \Delta$ then there exists an r such that $(q, r) \in \Delta$ and $(r, r') \in R$.*

Suppose that, before starting the reachability analysis of a transition system, we know that a certain equivalence relation $\approx$ is a bisimulation and respects the predicate ϕ in the sense that either all states in an equivalence class satisfy ϕ or none of them do. Then, when doing reachability analysis, it suffices to store and explore only a single element of each equivalence class. To implement the state space exploration, a *representative function* θ may be used that converts a state to a representative of the equivalence class of that state:

$$\forall_{q \in Q}\ (q \approx \theta(q)) \tag{1}$$

Using θ, we may improve the algorithm in Figure 1 by replacing lines 2 and 8, respectively, by:

$$(2) \quad \mathit{waiting} := \{\, \theta(q) \mid q \in Q_0 \,\}$$

$$(8) \quad \mathit{waiting} := \mathit{waiting} \cup \{\, \theta(q') \mid (q, q') \in \Delta \,\}$$

It can easily be shown that the adjusted algorithm remains correct: for all (finite) transition systems the outcomes of the original and the adjusted algorithm are equal. If the representative function is "good", which means that many equivalent states are projected onto the same representative, then the number of states to explore, and consequently the size of the *passed* set, may decrease dramatically. However, in order to apply the approach, the following two problems need to be solved:

- A suitable bisimulation equivalence $\approx$ that respects ϕ needs to be statically derived from the system description.
- An appropriate representative function θ needs to be constructed that satisfies equation (1), and that can be computed efficiently. Ideally, θ satisfies $q \approx q' \Rightarrow \theta(q) = \theta(q')$, in which case it is called *canonical*.

In this paper, we use symmetries to solve these problems. As in [21], the notion of *automorphism* is used to characterize symmetry within a transition system. This is a bijection on the set of states that (viewed as a relation) is a bisimulation. Phrased alternatively:

Definition 2.2 (Automorphism) *An* automorphism *on some transition system, say* (Q, Q_0, Δ)*, is a bijection* $h : Q \to Q$ *such that*

1. $q \in Q_0$ *if and only if* $h(q) \in Q_0$ *for all* $q \in Q$*, and*
2. $(q, q') \in \Delta$ *if and only if* $(h(q), h(q')) \in \Delta$ *for all* $q, q' \in Q$*.*

Let H be a set of automorphisms, let **id** be the identity function on states, and let $G(H)$ be the closure of $H \cup \{\mathbf{id}\}$ under inverse and composition. It can be shown that $G(H)$ is a group, and it induces a bisimulation equivalence relation $\approx$ on the set of states as follows:

$$q \approx q' \iff \exists_{h \in G(H)}\ (h(q) = q') \tag{2}$$

We introduce a symmetric data type to let the user explicitly point out the symmetries in the model. Simple static checks can ensure that the symmetry that is pointed out is not broken. Our approach to the second problem of coming up with good representative functions consists of "sorting the state" w.r.t. some ordering relation on states using the automorphisms. For instance, given a state q and a set of automorphisms, find the smallest state q' that can be obtained by repeatedly applying automorphisms and their inverses to q. It is clear that such a θ satisfies Equation 1, since it is constructed from the automorphisms only.

3 Adding Scalarsets to UPPAAL

The tool UPPAAL is a model checker for networks of timed automata extended with discrete variables (bounded integers, arrays) and blocking, binary synchronization as well as non-blocking broadcast communication (see for instance [23]). In the remainder of this section we illustrate by an example UPPAAL's description language extended with a *scalarset* type constructor allowing symmetric data types to be syntactically defined. Our extension is based on the notion of scalarset first introduced by Ip and Dill in the finite-state model checking tool MURφ [12, 21]. Also our extension is based on the C-like syntax to be introduced in the forthcoming version 4.0 of UPPAAL.

To illustrate our symmetry extension of UPPAAL we consider Fischer's mutual exclusion protocol. This protocol consists of n processes, identical up to their unique process identifiers. The purpose of the protocol is to ensure mutual exclusion on the critical sections of the processes. This is accomplished by letting each process write its identifier (`pid`) in a global variable (`id`) before entering its critical section. If after some given lower time bound (say 2) `id` still contains the `pid` of the process, then it may enter its critical section.

A scalarset of size n may be considered as the subrange $\{0, 1, \ldots, n-1\}$ of the natural numbers. Thus, the n process identifiers in the protocol can be modeled using a scalarset with size n. In addition to the global variable `id`, we use the array `active` to keep track of all active locations of the processes[1]. Global declarations are the following:

```
typedef scalarset[3] proc_id;  // a scalarset type with size 3
proc_id id;                    // declaration of a proc_id
                               //   variable
bool set;                      // declaration of a boolean
int active[proc_id];           // declaration of an array
                               //   indexed by proc_id
```

The first line defines `proc_id` to be a scalarset type of size 3, and the second line declares `id` to be a variable over this type. Thus `scalarset` is viewed as a type constructor. In the last line we show a declaration of an array indexed by elements of the scalarset `proc_id`.

At this point the only thing missing is the declaration of the actual processes in the system. In the description language of UPPAAL, processes are obtained as instances of

[1] This array is actually redundant and not present in the standard formulations of the protocol. It is, however, useful for showing important aspects of our extension.

parameterized process templates. In general, templates may contain several different parameters (e.g. bounded integers, clocks, and channels). In our extension we allow in addition the use of scalarsets as parameters. In the case of Fischer's protocol the processes of the system are given as instances of the template depicted in Figure 2.

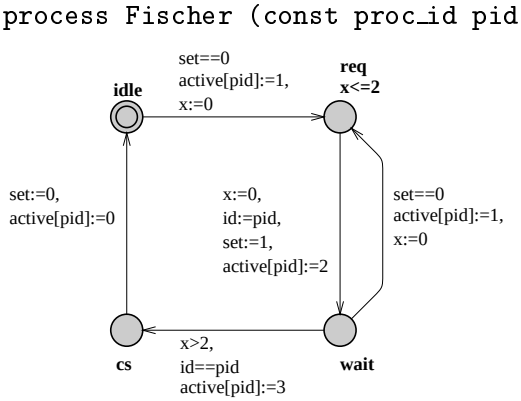

Figure 2: The template for Fischer's protocol.

The template has one local clock, `x`, and no local variables. Note that the header of the template defines a (constant) scalarset parameter `pid` of type `proc_id`. Access to the critical section `cs` is governed by suitable updates and tests of the global scalarset variable `id` together with upper and lower bound time constraints on when to proceed from requesting access (`req`) respectively proceed from waiting for access (`wait`). Note that all transitions update the array `active` to reflect the current active location of the process. The instantiation of this template and declaration of all three process in the system can be done as follows:

```
FischerProcs = forall i in proc_id : Fischer(i);
system FischerProcs;
```

The `forall` construct iterates over all elements of a declared scalarset type. In this case the iteration is over `proc_id` and a set of instances of the template `Fischer` is constructed and bound to `FischerProcs`. In the second line the final system is defined to be precisely this set.

4 Using Scalarsets for Symmetry Reduction

This section first presents a method to extract automorphisms from a UPPAAL system description using the new scalarset type. These automorphisms can be used for computation of the representative of a state as described in Section 2. Second, a total preorder is introduced on the individual clocks of zones that are generated during the exploration of the state space. Third, a representative function is defined that uses this preorder on clocks. The main technical result is a proof that this function is canonical under certain assumptions. The representative function may not be canonical without all these assumptions, but it certainly is sound.

4.1 Extraction of Automorphisms

The extended syntax as described in the previous section enables us to derive the following information from a system description:

1. A set Ω of scalarset types.
2. For each $\alpha \in \Omega$: (i) a set V_α of regular variables of type α[2], and (ii) a set D_α of pairs (a, n), where a is an integer variable or clock array and n is a dimension of a that is to be indexed by α elements such that $\alpha \neq \beta \Rightarrow D_\alpha \cap D_\beta = \emptyset$.
3. A partial mapping $\gamma : P \times \Omega \hookrightarrow \mathbb{N}$ that gives for each process p and scalarset α the element of α with which p is instantiated. This mapping is defined by quantification over scalarsets in the process definition section.

A UPPAAL state is a tuple $(\vec{l}, v, Z)$, where $\vec{l}$ is the location vector, v is the integer variable valuation, and Z is a *zone*. A zone is a set of *clock valuations*, i.e., functions $\nu : X \to \mathbb{R}_+$ where X is the set of clocks and $\mathbb{R}_+$ denotes the set of non-negative real numbers. Zones are represented in UPPAAL by *difference bounded matrices* (DBMs) [7, 11]. Concretely, the location vector and variable valuation are implemented by arrays of integers, and the DBM is implemented by a matrix of integers. The UPPAAL state representation assumes that every process has a fixed index in the location vector, every regular integer variable and every entry of an integer array variable has a fixed index in the variable valuation, and that every clock has a fixed index in the DBM. Thus, there are three injections, all denoted by ρ, that map processes, integer variables, and clocks to indices.

Next, we introduce the notion of *substate* for every scalarset element. Informally, the substate of element i of the scalarset α is a triple containing (i) indices of processes that have been instantiated with element i of α, (ii) indices of variables (or array entries) that are associated with element i of α, and (iii) indices of clocks that are associated with element i of α. These substates can statically be derived, and do not change during the state space exploration.

Definition 4.1 (Substate) *Let i be an element of the scalarset α. The substate of this element, denoted by substate(α, i), is a tuple $(\vec{l}, \vec{v}, \vec{c})$, where*

- *$\vec{l}$ is the* ordered *sequence of indices of all processes in $\{p \mid \gamma(p, \alpha) = i\}$.*
- *$\vec{v}$ is the* ordered *sequence of indices of*
  - *the local integer variables of all processes p that satisfy $\gamma(p, \alpha) = i$.*
  - *integer array entries that are associated with the i-th element of α, i.e., the entry $b[j_1] \cdots [j_k] \cdots [j_n]$ is associated with the i-th element of α if and only if $(b, k) \in D_\alpha$ and $j_k = i$.*

[2] The soundness proof in [16] uses the so-called $used_\alpha$ set for this set. Moreover, it is assumed that arrays of integer variables cannot be part of this $used_\alpha$ set, which is needed for the soundness proof of the state swaps. Since this soundness proof is reused in the present paper, the assumption that V_α only consists of regular variables should be made. This assumption, however, can probably be dropped.

- $\vec{c}$ *is the* ordered *sequence of indices of*
  - *local clocks of all processes p that satisfy $\gamma(p, \alpha) = i$.*
  - *clock array entries that are associated with the i-th element of α, i.e., the entry $c[j_1] \cdots [j_k] \cdots [j_n]$ is associated with the i-th element of α if and only if $(c, k) \in D_\alpha$ and $j_k = i$.*

We use these substates to define the automorphisms, and to this end we need the next assumptions. We note that the definition of automorphisms that appears in [16] does not need the assumptions below and therefore is more general, but also more complicated. Moreover, these assumptions are also needed in the proof that the representative function is canonical.

Assumption 4.2 (Basic assumptions)

(1) Local arrays are not *indexed by scalarsets and V_α only contains* global *variables.*

(2) At most one dimension of an array can be indexed by a scalarset, i.e., $|\{(b, n) \in D_\alpha \mid \alpha \in \Omega \wedge n \in \mathbb{N}\}| \leq 1$ for all arrays of integer and clock variables b.

(3) A process can be associated with at most one scalarset element, i.e., $|\{(\alpha, i) \mid \gamma(p, \alpha) = i \wedge \alpha \in \Omega\}| \leq 1$ for all processes p.

The contributions to the state of different substates are completely disjoint, which is formalized by the following lemma.

Lemma 4.3 *If $\alpha \neq \beta$ or $i \neq j$, then substate(α, i) and substate(β, j) are disjoint.*

PROOF. Let *substate*$(\alpha, i) = (\vec{l}_1, \vec{v}_1, \vec{c}_1)$ and let *substate*$(\beta, j) = (\vec{l}_2, \vec{v}_2, \vec{c}_2)$.

1. $\vec{l}_1$ and $\vec{l}_2$ share no indices. For the proof, assume that they do, i.e., by Definition 4.1 a process p exists such that $\gamma(p, \alpha) = i$ and $\gamma(p, \beta) = j$. We see, however, that p now is associated with two scalarset elements, since $\alpha \neq \beta$ or $i \neq j$, which contradicts the third item of Assumption 4.2. Therefore, $\vec{l}_1$ and $\vec{l}_2$ share no indices.

2. $\vec{v}_1$ and $\vec{v}_2$ share no indices. In the previous item we proved that the substates refer to disjoint sets of processes. Thus, the sets of local variables of these sets of processes also are disjoint. Now consider an array entry $b[h_1] \ldots [h_n]$ and assume that it is associated with the i-th element of α and with the j-th element of β. This means (see Definition 4.1) that some k exists such that $(b, k) \in D_\alpha \wedge h_k = i$ and some k' exists such that $(b, k') \in D_\beta \wedge h_{k'} = j$. From our assumption that $\alpha \neq \beta \vee i \neq j$ we can easily prove that $k \neq k'$: (1) if $\alpha \neq \beta$, then $D_\alpha \cap D_\beta = \emptyset$. Therefore, $(b, k) \neq (b, k')$ and clearly $k \neq k'$. (2) if $i \neq j$, then $h_k \neq h_{k'}$ and clearly $k \neq k'$. Thus, two dimensions of b are indexed by scalarsets which clearly contradicts the second item in Assumption 4.2. Therefore, $\vec{v}_1$ and $\vec{v}_2$ share no indices.

3. $\vec{c}_1$ and $\vec{c}_2$ share no indices. We can prove this by a similar argument as in the previous case.

■

Consider some substate $(\vec{l}_1, \vec{v}_1, \vec{c}_1)$ and a state $q = (\vec{l}, v, Z)$. We can *project* the state to this substate: $[\![\vec{l}_1]\!]_q$ is obtained from $\vec{l}_1$ by replacing every index by the encoding of the associated active location (according to q of course). The projection of q to $\vec{v}_1$, denoted by $[\![\vec{v}_1]\!]_q$, is obtained similarly. If we use the notation $[\vec{l}]_k$ to refer to the k-th element in the sequence $\vec{l}$, then for all k:

$$[\![\vec{l}_1]\!]_q = \vec{l}'_1 \quad \text{where } [\vec{l}'_1]_k = [\vec{l}]_{[\vec{l}_1]_k} \tag{3}$$

$$[\![\vec{v}_1]\!]_q = \vec{v}'_1 \quad \text{where } [\vec{v}'_1]_k = [v]_{[\vec{v}_1]_k} \tag{4}$$

Note that the location and variable projections can be ordered using the lexicographical order on sequences of numbers. We cannot easily define the projection of a state to the clock indices, since the state contains a *set* of clock valuations. For a single clock valuation ν, however, we define the projection of ν to a clock index vector $\vec{c}_1$ as follows:

$$[\![\vec{c}_1]\!]_\nu = \vec{c}'_1 \quad \text{where } [\vec{c}'_1]_k = \nu(\rho^{-1}([\vec{c}_1]_k)) \tag{5}$$

Clock valuation projections can be ordered using the lexicographical order on sequences of numbers. In the next subsection a preorder on clocks is defined that enables us to compare the projections of the clock parts of substates for any given state.

The next assumption formalizes the correspondence between substates of different elements of a scalarset. This correspondence is ensured by the implementation of UPPAAL, and is needed to define the state swaps that are used for the computation of representatives.

Assumption 4.4 *Consider substate$(\alpha, i) = (\vec{l}_1, \vec{v}_1, \vec{c}_1)$, and substate$(\alpha, j) = (\vec{l}_2, \vec{v}_2, \vec{c}_2)$.*

1. *The length of $\vec{l}_1$ equals the length of $\vec{l}_2$, and $i < j \Leftrightarrow [\vec{l}_1]_k < [\vec{l}_2]_k$.*
2. *The length of $\vec{v}_1$ equals the length of $\vec{v}_2$, and $i < j \Leftrightarrow [\vec{v}_1]_k < [\vec{v}_2]_k$, and $[\vec{v}_1]_k$ and $[\vec{v}_2]_k$ refer to equivalent variables:*
   - *$[\vec{v}_1]_k$ is the index of the local variable b of the process $\rho^{-1}([\vec{l}_1]_q)$ if and only if $[\vec{v}_2]_k$ is the index of the local variable b of the process $\rho^{-1}([\vec{l}_2]_q)$.*
   - *$[\vec{v}_1]_k$ is the index of the array entry $b[h_1]\dots[h_{p-1}][i][h_{p+1}]\dots[h_q]$ if and only if $[\vec{v}_2]_k$ is the index of $b[h_1]\dots[h_{p-1}][j][h_{p+1}]\dots[h_q]$.*
3. *The length of $\vec{c}_1$ equals the length of $\vec{c}_2$, and $i < j \Leftrightarrow [\vec{c}_1]_k < [\vec{c}_2]_k$, and $[\vec{c}_1]_k$ and $[\vec{c}_2]_k$ refer to equivalent clocks (defined as above).*

Assumptions 4.2 and 4.4 enable us to define so-called *state swaps*[3].

Definition 4.5 (State swap) *Consider two distinct elements, say i and j, of some scalarset α. Let substate$(\alpha, i) = (\vec{l}_1, \vec{v}_1, \vec{c}_1)$, and let substate$(\alpha, j) = (\vec{l}_2, \vec{v}_2, \vec{c}_2)$. The* state swap *is defined as $swap^\alpha_{i,j}((\vec{l}, v, Z)) = (\vec{l}', v', Z')$, where:*

[3] A more general definition that covers the current definition, but does not need Assumption 4.2 appears in [16]. E.g., that definition also covers the case when an array is indexed by multiple scalarsets.

- $\vec{l}'$ *is defined as follows for all* k*:*

$$[\vec{l}']_k = \begin{cases} [\vec{l}]_k & \text{if } k \notin \vec{l}_1 \text{ and } k \notin \vec{l}_2 \\ [\vec{l}]_m & \text{if } k = [\vec{l}_1]_n, \text{ where } m = [\vec{l}_2]_n \\ [\vec{l}]_m & \text{if } k = [\vec{l}_2]_n, \text{ where } m = [\vec{l}_1]_n \end{cases}$$

- v' *can be defined as follows (remember that* V_α *only contains global non-array variables):*

$$[v']_k = \begin{cases} [v]_k & \text{if } k \notin \vec{v}_1 \text{ and } k \notin \vec{v}_2 \text{ and } \rho^{-1}(k) \notin V_\alpha \\ i & \text{if } \rho^{-1}(k) \in V_\alpha \text{ and } [v]_k = j \\ j & \text{if } \rho^{-1}(k) \in V_\alpha \text{ and } [v]_k = i \\ [v]_m & \text{if } k = [\vec{v}_1]_n, \text{ where } m = [\vec{v}_2]_n \\ [v]_m & \text{if } k = [\vec{v}_2]_n, \text{ where } m = [\vec{v}_1]_n \end{cases}$$

- $Z' = \{s(\nu) \mid \nu \in Z\}$*, where the clock value swap* s *is defined for all clocks* x *as:*

$$(s(\nu))(x) = \begin{cases} \nu(x) & \text{if } \rho(x) \notin \{[\vec{c}_1]_k, [\vec{c}_2]_k\} \text{ for all } k \\ \nu(\rho^{-1}([\vec{c}_1]_k)) & \text{if } \rho(x) = [\vec{c}_2]_k \\ \nu(\rho^{-1}([\vec{c}_2]_k)) & \text{if } \rho(x) = [\vec{c}_1]_k \end{cases}$$

Note that by definition $swap^\alpha_{i,j}(q) = swap^\alpha_{j,i}(q)$. A number of syntactic checks have been identified in [16] that ensure that the symmetry suggested by the scalarsets is not broken. These checks are very similar to those originally identified for the Murφ verification system [21]. For instance, it is not allowed to use variables of a scalarset type for arithmetical operations such as addition. The next soundness theorem has been proven in [16] (provided that the symmetry is not broken)[4].

Theorem 4.6 (Soundness) *Every state swap is an automorphism.*

As a result, the representative function θ can be implemented by minimization of the state using the state swaps. Note that every state swap resembles a transposition of two scalarset elements. Hence, the equivalence classes induced by the state swaps originating from a scalarset with size n consist of at most $n!$ states. The maximal theoretical gain that can be achieved using the state swaps therefore is in the order of a factor $n!$.

Consider the instance of Fischer's mutual exclusion protocol as described in the previous section with three processes. There are three state swap functions: $swap^{\texttt{proc\_id}}_{0,1}$, $swap^{\texttt{proc\_id}}_{0,2}$ and $swap^{\texttt{proc\_id}}_{1,2}$. Now consider the following state of the model (the active location of the i-th process is given by $[\vec{l}]_i$ and the local clock of this process is given by x_i; also note that the zone Z only contains one clock valuation):

$$\begin{array}{lcl} \vec{l} & : & (\texttt{idle}, \texttt{wait}, \texttt{cs}) \\ v & : & \texttt{id} = 2,\ \texttt{set} = 1,\ \texttt{active}[0] = 0,\ \texttt{active}[1] = 2,\ \texttt{active}[2] = 3 \\ Z & : & x_0 = 4,\ x_1 = 3,\ x_2 = 2.5 \end{array}$$

[4] The soundness theorem has also been proven correct for the more general definition of the state swap function that appears in [16]. Thus, a definition of state swaps exists such that Theorem 4.6 holds without the need for Assumption 4.2.

When we apply $swap_{0,2}^{\texttt{proc\_id}}$ to this state, the result is the following state:

$$\begin{array}{lcl} \vec{l} & : & (\texttt{cs}, \texttt{wait}, \texttt{idle}) \\ v & : & \texttt{id} = 0,\ \texttt{set} = 1,\ \texttt{active}[0] = 3,\ \texttt{active}[1] = 2,\ \texttt{active}[2] = 0 \\ Z & : & x_0 = 2.5,\ x_1 = 3,\ x_2 = 4 \end{array}$$

The process swap swaps l_0 with l_2, and x_0 with x_2. Moreover, the value of the variable `id` is changed from 2 to 0, since $\texttt{id} \in V_{\texttt{proc\_id}}$, and the values of $\texttt{active}[0]$ and $\texttt{active}[2]$ are swapped. Applying $swap_{1,2}^{\texttt{proc\_id}}$ to this state gives the following state:

$$\begin{array}{lcl} \vec{l} & : & (\texttt{cs}, \texttt{idle}, \texttt{wait}) \\ v & : & \texttt{id} = 0,\ \texttt{set} = 1,\ \texttt{active}[0] = 3,\ \texttt{active}[1] = 0,\ \texttt{active}[2] = 2 \\ Z & : & x_0 = 2.5,\ x_1 = 4,\ x_2 = 3 \end{array}$$

Note that this swap does not change the value of `id`, since the scalarset elements 1 and 2 are interchanged and `id` contains scalarset element 0.

4.2 A Preorder on Clocks

The zone semantics seems to render a straightforward comparison of clocks impossible, since there are in general many different clock valuations in a zone. If we assume, however, that the UPPAAL model resets its clocks to zero only and that the convex-hull over-approximation is not used, then the zones that are generated by the forward state space exploration satisfy the *diagonal property*. This property informally states that a zone never contains valuations on both sides of a diagonal. This implies that the individual clocks can always be ordered using the order in which they were reset. To formalize this, three binary relations on the set of clocks X parameterized by a zone Z are defined:

$$x \preccurlyeq_Z y \iff \forall_{\nu \in Z}\ \nu(x) \leq \nu(y) \tag{6}$$

$$x \approx_Z y \iff \forall_{\nu \in Z}\ \nu(x) = \nu(y) \tag{7}$$

$$x \prec_Z y \iff (x \preccurlyeq_Z y \wedge x \not\approx_Z y) \tag{8}$$

Clearly, the relation $\preccurlyeq_Z$ is reflexive and transitive and hence it is a preorder on the set of clocks. Totality of this preorder w.r.t. zones that are generated during the state space exploration follows from the diagonal property.

Lemma 4.7 (Diagonal property) *Consider the state space exploration algorithm described in Figure 6 of [23][5]. Assume that the clocks are reset to the value 0 only and that the convex-hull over-approximation is not used. Then for all states $(\vec{l}, v, Z)$ stored in the waiting and passed list during a run of the algorithm and for all clocks x and y it holds that either $x \prec_Z y$, $x \approx_Z y$ or $y \prec_Z x$.*

PROOF. We prove that the diagonal property holds for the arbitrary clocks x and y by an inductive argument. Consider the initial zone, which contains only one clock valuation. It is clear that the diagonal property holds for such a zone. Before we prove

[5] Essentially, this is a UPPAAL tailored instance of the algorithm in Figure 1 of the present paper.

the induction step we observe that $x \approx_Z y$, $x \prec_Z y$, and $y \prec_Z x$ are mutually exclusive: if one holds then the remaining two do not hold. Now consider a zone Z that satisfies the diagonal property. If the convex-hull over-approximation is not used, then the three operations on zones that are used during state space exploration are the following [2]:

1. Intersection with zone Z'. This results in a zone $Z'' \subseteq Z$. Now assume that $x \prec_Z y$, which means that $\nu(x) \leq \nu(y)$ for all $\nu \in Z$, and that a $\nu \in Z$ exists such that $\nu(x) \neq \nu(y)$. Clearly, for all $\nu'' \in Z''$ still holds that $\nu''(x) \leq \nu''(y)$. Next, we distinguish two cases: First, a $\nu'' \in Z''$ exists such that $\nu''(x) \neq \nu''(y)$. Then clearly $x \prec_{Z''} y$. Second, such a ν'' does not exist. Then $\nu''(x) = \nu''(y)$ for all $\nu'' \in Z''$ and thus $x \approx_{Z''} y$. It is straightforward to see that $x \approx_Z y$ implies that $x \approx_{Z''} y$. Thus, the diagonal property also holds for Z''.

2. Resetting clock x in zone Z. This results in the zone $Z' = \{\nu[x := 0] \,|\, \nu \in Z\}$, where $\nu[x := 0](u) = \nu(u)$ if $u \neq x$, and $\nu[x := 0](x) = 0$. We distinguish two cases:

   - $u \sim_Z w$, where $u \neq x$ and $w \neq x$. Thus, the clock valuations in Z' have not changed for u and w, and clearly $u \sim_{Z'} w$.
   - $x \sim_Z y$. If there exists a $\nu' \in Z'$ such that $\nu'(y) > 0$, then $x \prec_{Z'} y$ by definition, since $\nu'(x) = 0$ and $\nu'(y) \geq 0$ for all $\nu' \in Z'$. Otherwise, $\nu'(x) = \nu'(y) = 0$ for all $\nu' \in Z'$, and hence $x \approx_{Z'} y$. (Note that resetting clocks to values greater than zero destroys the property for this case.)

   Thus, the diagonal property also holds for Z'.

3. Time elapse. This removes the upper bounds on the individual clocks: $Z' = \{\nu + \delta \,|\, \nu \in Z \wedge \delta \in \mathbb{R}_+\}$, where $(\nu + \delta)(x) = \nu(x) + \delta$ for all $x \in X$. Now assume that $x \prec_Z y$, which means that $\nu(x) \leq \nu(y)$ for all $\nu \in Z$, and that a $\nu \in Z$ exists such that $\nu(x) \neq \nu(y)$. First, consider a $(\nu + \delta) \in Z'$. Clearly, $(\nu + \delta)(x) \leq (\nu + \delta)(y)$, since $\nu(x) \leq \nu(y)$. Second, consider the $\nu \in Z$ such that $\nu(x) \neq \nu(y)$. By definition, $(\nu + \delta) \in Z'$ for any δ. Clearly, $(\nu + \delta)(x) \neq (\nu + \delta)(y)$. Therefore, $x \prec_{Z'} y$. Now assume that $x \approx_Z y$, which means that $\nu(x) = \nu(y)$ for all $\nu \in Z$. By definition, $(\nu + \delta)(x) = (\nu + \delta)(y)$, and clearly $x \approx_{Z'} y$. Thus, the diagonal property also holds for Z'.

(Note that the convex-hull over-approximation uses the union of zones which clearly does not preserve the diagonal property.) This proves that every zone generated during the state space exploration satisfies the diagonal property. ■

The clock parts of two substates can be compared using a lexicographical preorder that has been based on the $\preccurlyeq_Z$ preorder.

Definition 4.8 (Clock preorder) *Let $\vec{c}_1$ and $\vec{c}_2$ be two clock index vectors with length k, and let q be a state with zone Z. We say that $\vec{c}_1 <_q \vec{c}_2$ if and only if*

$$\exists_{0 \leq i < k} \ \left(\rho^{-1}([\vec{c}_1]_i) \prec_Z \rho^{-1}([\vec{c}_2]_i) \ \wedge \ \forall_{0 \leq j < i} \ \left(\rho^{-1}([\vec{c}_1]_j) \approx_Z \rho^{-1}([\vec{c}_2]_j)\right)\right)$$

The non-strict version of the clock order is defined as usual: $\vec{c}_1 \leq_q \vec{c}_2$ if and only if $\vec{c}_1 <_q \vec{c}_2$ or $\rho^{-1}([\vec{c}_1]_j) \approx_Z \rho^{-1}([\vec{c}_2]_j)$ for all $0 \leq j < k$.

Lemma 4.9 *If the clocks in the model are reset to zero only and the convex hull over-approximation is not used, then the relation on clock index vectors of equal length as defined in Definition 4.8 is a total preorder.*

PROOF. Straightforward, since $\preccurlyeq_Z$ is a total preorder on the set of clocks under the mentioned premises (see Lemma 4.7). ■

The next lemma relates the clock preorder that is defined for zones to the projections of the state to the individual clock valuations.

Lemma 4.10 *Let $\vec{c}_1$ and $\vec{c}_2$ be two clock index vectors with length k and let q be a state with zone Z. If some $\nu \in Z$ exists such that $[\![\vec{c}_1]\!]_\nu < [\![\vec{c}_2]\!]_\nu$, then $\vec{c}_1 <_q \vec{c}_2$.*

PROOF. Assume that a $\nu \in Z$ exists such that $[\![\vec{c}_1]\!]_\nu < [\![\vec{c}_2]\!]_\nu$. From Equation 5 we know that a $0 \leq j < k$ exists such that $\nu(\rho^{-1}([\vec{c}_1]_j)) < \nu(\rho^{-1}([\vec{c}_2]_j))$ and $\nu(\rho^{-1}([\vec{c}_1]_i)) = \nu(\rho^{-1}([\vec{c}_2]_i))$ for all $0 \leq i < j$. Now suppose that $\vec{c}_1 \not<_q \vec{c}_2$. Since the preorder is total (see Lemma 4.9), we consider the two remaining possibilities. First, suppose that $\vec{c}_1 =_q \vec{c}_2$. By Definition 4.8 and Equations 6–8: for all $\nu \in Z$ must hold that $\nu(\rho^{-1}([\vec{c}_1]_j)) = \nu(\rho^{-1}([\vec{c}_2]_j))$ for all $0 \leq j < k$. This clearly does not hold, and from this contradiction we can conclude that $\vec{c}_1 \neq_q \vec{c}_2$. Second, suppose that $\vec{c}_1 >_q \vec{c}_2$. By Definition 4.8 and Equations 6–8: for all $\nu \in Z$ must hold that $\nu(\rho^{-1}([\vec{c}_1]_j)) \geq \nu(\rho^{-1}([\vec{c}_2]_j))$ for all $0 \leq j < k$. This clearly also does not hold, and we conclude that $\vec{c}_1 \not>_q \vec{c}_2$. Thus, $\vec{c}_1 <_q \vec{c}_2$. ■

In the next subsection we define a total preorder on substates and use the state swaps to compute the representative of a symmetry class, which under certain assumptions is canonical.

4.3 Computation of Representatives

A comparison between the state contributions of different scalarset elements is defined as follows.

Definition 4.11 (Substate preorder) *Consider substate$(\alpha, i) = (\vec{l}_1, \vec{v}_1, \vec{c}_1)$, and also substate$(\alpha, j) = (\vec{l}_2, \vec{v}_2, \vec{c}_2)$, and let q be a state. Then substate$(\alpha, i) <_q$ substate(α, j) iff*

- $[\![\vec{l}_1]\!]_q < [\![\vec{l}_2]\!]_q$ *or*
- $[\![\vec{l}_1]\!]_q = [\![\vec{l}_2]\!]_q$ *and* $[\![\vec{v}_1]\!]_q < [\![\vec{v}_2]\!]_q$, *or*
- $[\![\vec{l}_1]\!]_q = [\![\vec{l}_2]\!]_q$ *and* $[\![\vec{v}_1]\!]_q = [\![\vec{v}_2]\!]_q$ *and* $\vec{c}_1 <_q \vec{c}_2$.

The non-strict version is defined as usual: substate$(\alpha, i) \leq_q$ substate(α, j) if and only if substate$(\alpha, i) <_q$ substate(α, j) or $[\![\vec{l}_1]\!]_q = [\![\vec{l}_2]\!]_q \wedge [\![\vec{v}_1]\!]_q = [\![\vec{v}_2]\!]_q \wedge \vec{c}_1 =_q \vec{c}_2$.

Lemma 4.12 *If the clocks in the model are reset to zero only and the convex-hull over-approximation is not used, then the relation as defined in Definition 4.11 is a total preorder on the substates of a scalarset.*

PROOF. Straightforward since the preorders on the three components of the substate are total under the mentioned assumptions. ■

The next lemma states how the substate preorder is affected by state swaps.

Lemma 4.13 *Let $q' = swap^{\alpha}_{i,j}(q)$, and let $\pi(i) = j$, $\pi(j) = i$, and $\pi(k) = k$ for all $k \neq i, j$. Assume that $substate(\beta, m) \sim_q substate(\beta, n)$, where $\sim \in \{<, =, >\}$.*

- *If $\alpha \neq \beta$, then $substate(\beta, m) \sim_{q'} substate(\beta, n)$.*
- *If $\alpha = \beta$, then $substate(\beta, \pi(m)) \sim_{q'} substate(\beta, \pi(n))$.*

PROOF. The first case can easily be proven using Lemma 4.3. The state swap of α does not affect the parts of the state that are relevant for the substates of β. Therefore, the order of the elements of β is not disturbed. For the second case assume that $\alpha = \beta$ and let $q = (\vec{l}, v, Z)$, $q' = (\vec{l'}, v', Z')$, $s_1 = substate(\beta, m) = (\vec{l}_1, \vec{v}_1, \vec{c}_1)$, and $s_2 = substate(\beta, n) = (\vec{l}_2, \vec{v}_2, \vec{c}_2)$.

- Suppose that $m = n$. Since $substate(\beta, k) =_q substate(\beta, k)$ for all states q and for all $\beta \in \Omega$ and $k \in \beta$ the lemma clearly holds.
- Suppose that i and j are not equal to m or n. Thus, $\pi(m) = m$ and $\pi(n) = n$. This case can easily be proven using Lemma 4.3 and Definition 4.5: the state swap does not affect the projections to the substates of m and n.
- Suppose that $m = i$ and $n \neq i, j$. Thus, $\pi(m) = j$ and $\pi(n) = n$. Let $s_3 = substate(\beta, j) = (\vec{l}_3, \vec{v}_3, \vec{c}_3)$. We prove that $[\![\vec{l}_3]\!]_{q'} = [\![\vec{l}_1]\!]_q$, $[\![\vec{v}_3]\!]_{q'} = [\![\vec{v}_1]\!]_q$, and that if $\vec{c}_1 \sim_q \vec{c}_2$, then $\vec{c}_3 \sim_{q'} \vec{c}_2$, where $\sim \in \{<, =, >\}$. This proves (see Definition 4.11) that $s_3 \sim_{q'} s_2$.

  1. Let $\vec{l}_1 = (l_1^0, l_1^1, \ldots, l_1^n)$ and let $\vec{l}_3 = (l_3^0, l_3^1, \ldots, l_3^n)$. By Definition 4.5 we see that $[\vec{l'}]_k = [\vec{l}]_k$ if $k \notin \vec{l}_1$ and $k \notin \vec{l}_2$. Moreover, the definition has the effect that the values of the entries at indices l_1^i and l_3^i are swapped for all $0 \leq i \leq n$. Clearly, $[\![\vec{l}_3]\!]_{q'} = [\![\vec{l}_1]\!]_q$.
  2. Since we assumed that V_α only contains global non-array variables (see Assumption 4.2 and the second assumption at the beginning of Section 4.1), we can use the same argument as in the previous item to prove $[\![\vec{v}_3]\!]_{q'} = [\![\vec{v}_1]\!]_q$.
  3. Assume that $\vec{c}_1 <_q \vec{c}_2$ and let $\vec{c}_1 = (c_1^0, c_1^1 \ldots, c_1^k)$, and $\vec{c}_2 = (c_2^0, c_2^1, \ldots, c_2^k)$. By Definition 4.8 we know that some $0 \leq f \leq k$ exists such that:

$$\rho^{-1}(c_1^f) \prec_Z \rho^{-1}(c_2^f) \tag{9}$$

$$\forall_{0 \leq e < f}\ \rho^{-1}(c_1^e) \approx_Z \rho^{-1}(c_2^e) \tag{10}$$

  This means by definition of the $\approx_Z$ and $\prec_Z$ relations that

$$\exists_{\nu \in Z}\ \nu(\rho^{-1}(c_1^f)) < \nu(\rho^{-1}(c_2^f)) \tag{11}$$

$$\forall_{0 \leq e < f}\ \forall_{\nu \in Z}\ \nu(\rho^{-1}(c_1^e)) = \nu(\rho^{-1}(c_2^e)) \tag{12}$$

Next, we apply Definition 4.5 which swaps the values of $\rho^{-1}(c_1^l)$ and $\rho^{-1}(c_3^l)$ for all $0 \leq l \leq n$ and lets the other clocks unchanged. Since substates are disjoint (see Lemma 4.3) we know that the clocks in $\vec{c}_2$ keep their values: $s(\nu)(\rho^{-1}(c_2^l)) = \nu(\rho^{-1}(c_2^l))$ for all $0 \leq l \leq n$. Furthermore, $s(\nu)(\rho^{-1}(c_1^l)) = \nu(\rho^{-1}(c_3^l))$ for all $v \in Z$ and $0 \leq l \leq n$. Combination with the previous equations gives us that:

$$\exists_{\nu \in Z'}\ \nu(\rho^{-1}(c_3^f)) < \nu(\rho^{-1}(c_2^f)) \tag{13}$$

$$\forall_{0 \leq e < f}\ \forall_{\nu \in Z'}\ \nu(\rho^{-1}(c_3^e)) = \nu(\rho^{-1}(c_2^e)) \tag{14}$$

By definition of the $\approx$ and $\prec$ relations:

$$\rho^{-1}(c_3^f) \prec_{Z'} \rho^{-1}(c_2^f) \tag{15}$$

$$\forall_{0 \leq e < f}\ \rho^{-1}(c_3^e) \approx_{Z'} \rho^{-1}(c_2^e) \tag{16}$$

By Definition 4.8 we can conclude that $\vec{c}_3 <_{q'} \vec{c}_2$. The case for $\vec{c}_1 =_q \vec{c}_2$ is similar.

- Suppose that $m = i$ and $n = j$. Thus, $\pi(m) = j$ and $\pi(n) = i$. With a similar argument as in the previous item we can prove that $[\![\vec{l}_1]\!]_{q'} = [\![\vec{l}_2]\!]_q$, $[\![\vec{l}_2]\!]_{q'} = [\![\vec{l}_1]\!]_q$, $[\![\vec{v}_1]\!]_{q'} = [\![\vec{v}_2]\!]_q$, $[\![\vec{v}_2]\!]_{q'} = [\![\vec{v}_1]\!]_q$, and if $\vec{c}_1 \sim_q \vec{c}_2$, then $\vec{c}_2 \sim_{q'} \vec{c}_1$, where $\sim \in \{<, =, >\}$. Hence, $s_2 \sim_{q'} s_1$.

■

We minimize the state by sorting the substate contributions of each scalarset according to the substate preorder of Definition 4.11. To this end, we apply a variation of the bubble-sort algorithm, see Figure 3. It is clear that this representative computation satisfies Equation 1 which ensures soundness, since states are transformed using the state swaps only, which are automorphisms by Theorem 4.6.

```
(1)   for all α ∈ Ω do
(2)     for i = 1 to |α| do
(3)       for j = |α| − 1 to i do
(4)         if substate(α, j) <_q substate(α, j − 1) then
(5)           q := swap^α_{j−1,j}(q)
(6)         {substate(α, j − 1) ≤_q substate(α, m), j ≤ m < |α|}
(7)       od
(8)       {substate(α, 0) ≤_q ··· ≤_q substate(α, i − 1)}
(9)     od
(10)    {m ≤ n ⇒ substate(α, m) ≤_q substate(α, n)}
(11)  od
```

Figure 3: Minimization of state q using the bubble-sort algorithm. The size of scalarset type α is denoted by $|\alpha|$. Lines 6, 8 and 10 show the loop invariants.

The following theorem states the main technical contribution of our work. Informally, it means that the detected symmetries are optimally used.

Theorem 4.14 (Canonical representative) *If Assumptions 4.2 and 4.4 are true, there are no variables of a scalarset type ($V_\alpha = \emptyset$), the convex-hull over-approximation is not used and the clocks in the model are reset to zero only, then the representative function θ as computed by the algorithm in Figure 3 is canonical.*

PROOF. The loop invariants can easily be proved using the fact that the preorder on substates is total (Lemma 4.12) and the fact that swapping two elements of a scalarset only has an effect on those two elements, i.e., the relations between other substates are not disturbed (Lemma 4.13).

Next, we prove that the algorithm computes a canonical representative, i.e., θ satisfies $q \approx q' \Rightarrow \theta(q) = \theta(q')$. Suppose that $q \approx q'$, i.e., a sequence of state swaps exists that transforms q into q'. Now consider $\theta(q) = (\vec{l}, v, Z)$ and $\theta(q') = (\vec{l'}, v', Z')$ (clearly, $\theta(q) \approx \theta(q')$ by Equation 1) and assume that they are different.

1. Assume that $\vec{l} \neq \vec{l'}$, more precisely, $[\vec{l}]_k \neq [\vec{l'}]_k$, and $[\vec{l}]_j = [\vec{l'}]_j$ for all $1 \leq j < k$. Without loss of generality we can also assume that $[\vec{l}]_k > [\vec{l'}]_k$. Clearly, some scalarset α and $i \in \alpha$ exist such that $k \in \vec{l_i}$, where $\vec{l_i} = [substate(\alpha, i)]_0$ since otherwise entry k cannot be swapped and as a result $\theta(q) \not\approx \theta(q')$ which we assumed. Moreover, exactly one such a combination exists, since substates are disjoint according to Lemma 4.3. Say that $\vec{l_i} = (l_i^0, l_i^1, \ldots, l_i^n)$ and that $l_i^h = k$. By Definition 4.1 we know that $l_i^g < l_i^h$ for all $0 \leq g < h$. Combination with the fact that $[\vec{l}]_j = [\vec{l'}]_j$ for all $1 \leq j < k$ gives us then that $[\![\vec{l_i}]\!]_{\theta(q)} > [\![\vec{l_i}]\!]_{\theta(q')}$.
   Since $\theta(q) \approx \theta(q')$ it is possible to transform $\theta(q)$ into $\theta(q')$ by state swaps. By Definition 4.5 and the fact that substates are disjoint (see Lemma 4.3), the i-th element of α can only be replaced by the following elements of α as a result of the transformation:

$$J = \{\, j \mid [\![\vec{l_j}]\!]_{\theta(q)} = [\![\vec{l_i}]\!]_{\theta(q')} \text{ where } \vec{l_j} = [substate(\alpha, j)]_0 \text{ and } 0 \leq j < |\alpha| \,\}$$

   Clearly, $J \neq \emptyset$ since that would mean that $\theta(q) \not\approx \theta(q')$. Now we show that a $j \in J$ must exist such that $j > i$. Therefore, assume that $j \leq i$ for all $j \in J$, and consider the following set of location vector indices which are exactly those indices whose entries in $\vec{l}$ can replace the value $[\vec{l}]_k$ as a result of the transformation that proves that $\theta(q) \approx \theta(q')$:

$$G = \{\, [\vec{l_j}]_h \mid \text{where } \vec{l_j} = [substate(\alpha, j)]_0 \text{ and } j \in J \,\}$$

   By Assumption 4.4 and the fact that $j \leq i$ (and $j \neq i$) for all $j \in J$ we know that $g < k$ for all $g \in G$. Thus, $[\vec{l}]_g = [\vec{l'}]_g$ for all $g \in G$ and $[\vec{l}]_k \neq [\vec{l'}]_k$. Clearly, $\vec{l}$ can never be transformed into $\vec{l'}$. This contradicts our assumption that $\theta(q) \approx \theta(q')$, and therefore we can conclude that a $j \in J$ exists such that $j > i$. Now we fix this $j > i$ and have that $[\![\vec{l_j}]\!]_{\theta(q)} = [\![\vec{l_i}]\!]_{\theta(q')}$. Above we have shown that $[\![\vec{l_i}]\!]_{\theta(q)} > [\![\vec{l_i}]\!]_{\theta(q')}$. Combination gives us that $i < j$ and $[\![\vec{l_i}]\!]_{\theta(q)} > [\![\vec{l_j}]\!]_{\theta(q)}$. In other words, $i < j$ and $substate(\alpha, i) >_{\theta(q)} substate(\alpha, j)$, which clearly contradicts the loop invariant in line 10 of the algorithm in Figure 3. Therefore, $\vec{l} = \vec{l'}$.

2. Assume that $\vec{l} = \vec{l'} \wedge v \neq v'$. Since we assumed that $V_\alpha = \emptyset$ we know that the difference between v and v' is in the projections to some substate. Therefore, the proof is the same as in the previous item.

3. Assume that $\vec{l} = \vec{l'} \wedge \vec{v} = \vec{v'} \wedge Z \neq Z'$. This means that a $\nu \in Z$ exists such that $\nu \notin Z'$. Moreover, since $\theta(q) \approx \theta(q')$, a $\nu' \in Z'$ exists such that ν can be transformed into ν' by the state swaps. Let us consider this ν and this ν'.
   By our assumptions, a clock index k exists such that $\nu(\rho^{-1}(k)) \neq \nu'(\rho^{-1}(k))$ and $\nu(\rho^{-1}(i)) = \nu'(\rho^{-1}(i))$ for all $0 \leq i \leq k$. Without loss of generality we can assume that $\nu(\rho^{-1}(k)) > \nu'(\rho^{-1}(k))$, and that $k \in \vec{c}_i = [substate(\alpha, i)]_2$. Thus, $[\![\vec{c}_i]\!]_\nu > [\![\vec{c}_i]\!]_{\nu'}$. With a similar argument as in the first item we can prove that a $j > i$ exists such that $[\![\vec{c}_j]\!]_\nu = [\![\vec{c}_i]\!]_{\nu'}$, where $\vec{c}_j = [substate(\alpha, j)]_2$.
   Combination gives us that $i < j$ and $[\![\vec{c}_i]\!]_\nu > [\![\vec{c}_j]\!]_\nu$. Applying Lemma 4.10 gives us then that $i < j$ and $\vec{c}_i >_{\theta(q)} \vec{c}_j$, which clearly contradicts the loop invariant in line 10 of the algorithm in Figure 3. Therefore, $Z = Z'$.

■

5 Experimental Results

This section presents and discusses experimental data that has been obtained with the UPPAAL prototype. The measurements were done using the tool *memtime*, for which a link can be found at the UPPAAL website *http://www.uppaal.com/*.

In order to demonstrate the effectiveness of symmetry reduction, the resource requirements for checking the correctness of Fischer's mutual exclusion protocol were measured as a function of the number of processes for both regular UPPAAL and the prototype, see Figure 4. A conservative extrapolation of the data shows that the verification of the protocol for 20 processes without symmetry reduction would take 115 days and 1000 GB of memory, whereas this verification can be done within approximately one second using less than 10 MB of memory with symmetry reduction.

Similar results have been obtained for the CSMA/CD protocol ([26, 30]) and for the timeout task of a distributed agreement algorithm[6] which is described in [5]. To be more precise, regular UPPAAL's limit for the CSMA/CD protocol is approximately ten processes, while the prototype can easily handle fifty processes. Similarly, the prototype can easily handle thirty processes for the model of the timeout task, whereas regular UPPAAL can only handle six processes.

Besides the three models discussed above, we also investigated the gain of symmetry reduction for two more complex models. First, we measured the gain for the previously mentioned agreement algorithm, of which we are unable to verify an interesting instance even with symmetry reduction due to the size of the state space. Nevertheless, symmetry reduction showed a very significant improvement for less interesting instances of the algorithm (only two symmetric processes). Second, we measured the gain for a model of Bang & Olufsen's audio/video protocol, which is described in [15]. This paper describes how UPPAAL is used to find an error in the protocol, and it describes the verification

[6] A UPPAAL model is available at *http://www.cs.kun.nl/ita/publications/papers/martijnh/*.

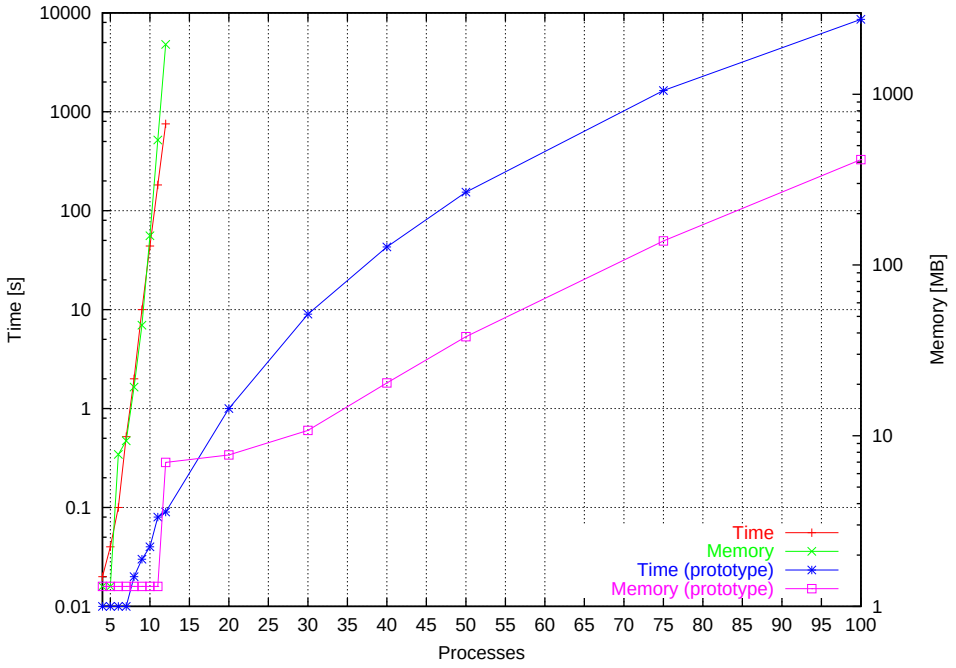

Figure 4: Run-time data for Fischer's mutual exclusion protocol showing the enormous gain of symmetry reduction. The step in the graph of the memory usage is probably due to the the fact that UPPAAL allocates memory in chunks of a few megabyte at a time.

of the corrected protocol for two (symmetric) senders. Naturally, we added another sender – verification of the model for three senders was impossible at the time of the first verification – and we found another error, whose source and implications we are investigating at the time of this writing. Table 1 shows run-time data for these models.

Table 1: Comparing the time and memory consumption of the relations for the agreement algorithm and for Bang & Olufsen's audio/video protocol with two and three senders. The exact parameters of the agreement model are the following: $n = 2$, $f = 1$, $ones = 0$, $c_1 = 1$, $c_2 = 2$ and d varied (the value is written between the parenthesis). Furthermore, the measurements were done for the verification of the agreement invariant only. Three verification runs were measured for each model and the best one w.r.t. time is shown.

Model	**Time** [s]		**Memory** [MB]	
	No red.	Red.	No red.	Red.
Agreement (0)	1	3	33	45
Agreement (1)	21	16	294	180
Agreement (2)	80	23	905	245
Agreement (3)	231	32	2126	321
B&O (2)	2	1	16	10
B&O (3)	265	36	1109	181

6 Conclusions

The results we obtained with our prototype are clearly quite promising: with relatively limited changes/extensions of the UPPAAL code we obtain a rather drastic improvement of performance for systems with symmetry that can be expressed using scalarsets.

An obvious next step is to do experiments concerning profiling where computation time is spent, and in particular how much time is spent on computing representatives. In the tool Design/CPN [20, 22, 13] (where symmetry reduction is a main reduction mechanism) there have been interesting prototype experiments with an implementation in which the (expensive) computations of representatives were launched as tasks to be solved in parallel with the main exploration algorithm.

Due to the presence of the global variable `id`, which has scalarset type, our model of Fischer's protocol does not satisfy the conditions of Theorem 4.14. And indeed the representative function θ as computed by the algorithm in Figure 3 is not fully canonical for this model. This is due to the fact that two processes can take the transition to location `wait` at the same moment. The projections to the resulting substates of the processes then are equal, but the value of `id` depends on the order of arrival. Our algorithm cannot distinguish these two different states. We claim, however, that the implementation does compute a canonical representative, since it also considers the V_α variables for the decision whether to swap two scalarset elements. Nevertheless, of course, it remains an interesting topic for future research to optimize the representative function for timed automata models that do not satisfy the restrictions of Theorem 4.14.

In this paper, we have exploited symmetries to statically derive bisimulations and (efficient) representative functions from system descriptions. A complementary static analysis technique for deriving bisimulations and representative functions is the *dead variable reduction* technique described in the PhD thesis of Karen Yorav [29]. In Yorav's terminology, a variable v is *used* in a transition $l \stackrel{g,a,up}{\longrightarrow} l'$ if v appears in g or in the right hand side of an assignment in up. Variable v is *defined* in the transition if it is in the left hand side of an assignment in up. Notice that in an assignment "$v := v+1$" v is first used, and then it is defined. A variable v is said to be *dead* at location l if on every execution path from l, v is defined before it is used, or is never used at all. Clearly, states that only differ on the values of dead variables are bisimilar, and any function that assigns a fixed value to these variables will give us a canonical representative function. An example of a dead variable is the global variable `id` in Fischer's protocol, of which the value does not matter for locations in which none of the components is in its waiting location. Dead variable reduction is closely related to the static guard analysis technique for timed automata as described in [6]. It would interesting to implement dead variable reduction in UPPAAL and to investigate the resulting speedup on some benchmark examples.

The scalarset approach that we follow in this paper only allows one to express total symmetries. An obvious direction for future research will be to study how other types of symmetry (for instance as we see it in a token ring) can be exploited.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[2] R. Alur. Timed automata. In *11th International Conference on Computer Aided Verification*, number 1633 in LNCS, pages 8–22. Springer–Verlag, 1999.

[3] R. Alur, C. Courcoubetis, and D. L. Dill. Model checking in dense real time. *Information and Computation*, 104:2–34, 1993.

[4] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *17th International Colloquium on Automata, Languages, and Programming*, pages 322–335, 1990.

[5] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 41(1):122–152, 1994.

[6] G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In H. Garavel and J. Hatcliff, editors, *TACAS 2003*, number 2619 in LNCS, pages 254–270. Springer–Verlag, 2003.

[7] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[8] D. Bosnacki, D. Dams, and L. Holenderski. A heuristic for symmetry reductions with scalarsets. In J.N. Oliveira and P. Zave, editors, *FME 2001*, number 2021 in LNCS, pages 518–533. Springer–Verlag, 2001.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[10] 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.

[11] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proc. of Automatic Verification Methods for Finite State Systems*, number 407 in LNCS, pages 197–212. Springer–Verlag, 1989.

[12] D. L. Dill, A. J. Drexler, A. J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society, 1992.

[13] L. Elgaard. *The Symmetry Method for Coloured Petri Nets - Theory, Tools, and Practical Use*. PhD thesis, Department of Computing Science, University of Aarhus, Denmark, July 2002.

[14] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In *CAV'93*, number 697 in LNCS. Springer–Verlag, 1993.

[15] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *18th IEEE Real-Time Systems Symposium*, pages 2–13, 1997.

[16] M. Hendriks. Enhancing UPPAAL by exploiting symmetry. Technical Report NIII-R0208, NIII, University of Nijmegen, October 2002.

[17] M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding symmetry reduction to UPPAAL. In *Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, 2004. To appear.

[18] T. A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[19] G. J. Holzmann. The SPIN model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[20] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Reachability trees for high-level Petri nets. *Theoretical Computer Science*, 45(3):261–292, 1986.

[21] C. N. Ip and D. L. Dill. Better verification through symmetry. In D. Agnew, L. Claesen, and R. Camposano, editors, *Computer Hardware Description Languages and their Applications*, pages 87–100, Ottawa, Canada, 1993. Elsevier Science Publishers B.V., Amsterdam, The Netherlands. Journal version appeared in *Formal Methods in System Design*, 9(1/2):41–75, 1996.

[22] K. Jensen. Condensed state spaces for symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9(1/2):7–40, 1996.

[23] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1/2):134–152, 1997.

[24] K. L. McMillan. *Symbolic Model Checking.* PhD thesis, Carnegie Mellon University, Pittsburgh, May 1992.

[25] P. H. Starke. Reachability analysis of Petri nets using symmetries. *Syst. Anal. Model. Simul./5*, 8(4):293–303, 1991.

[26] A. S. Tanenbaum. *Computer Networks.* Prentice–Hall, 1996.

[27] F. Wang. Efficient data structure for fully symbolic verification of real-time software systems. In S. Graf and M. Schwartzbach, editors, *TACAS'00*, number 1785 in LNCS, pages 157–171. Springer–Verlag, 2000.

[28] F. Wang and K. Schmidt. Symmetric symbolic safety-analysis of concurrent software with pointer data structures. In D.A. Peled and M.Y. Vardi, editors, *FORTE'02*, number 2529 in LNCS, pages 50–64. Springer–Verlag, 2002.

[29] K. Yorav. *Exploiting Syntactic Structure for Automatic Verification.* PhD thesis, The Technion, Israel Institute of Technology, Israel, June 2000.

[30] S. Yovine. KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1/2):123–133, 1997.