Abstract. The exponential explosion of parallel interleavings remains a fundamental challenge to model checking of concurrent programs. Both partial-order reduction (POR) and transaction reduction (TR) decrease the number of interleavings in a concurrent system. Unlike POR, transactions also reduce the number of intermediate states. Modern POR techniques, on the other hand, offer more dynamic ways of identifying commutative behavior, a crucial task for obtaining good reductions.

We show that transaction reduction can use the same dynamic commutativity as found in stubborn set POR. We also compare reductions obtained by POR and TR, demonstrating with several examples that these techniques complement each other. With an implementation of the dynamic transactions in the model checker LTSmin, we compare its effectiveness with the original static TR and two POR approaches. Several inputs, including realistic case studies, demonstrate that the new dynamic TR can surpass POR in practice.

1 Introduction

POR \cite{34,48,20} yields state space reductions by selecting a subset $P_\sigma$ of the enabled actions $E_\sigma$ at each state $\sigma$; the other enabled actions $E_\sigma \setminus P_\sigma$ are pruned. For instance, reductions preserving deadlocks (states without outgoing transitions) can be obtained by ensuring the following properties for the set $P_\sigma \subseteq E_\sigma \subseteq A$, where $A$ is the set of all actions:

- In any state $\sigma_n$ reachable from $\sigma$ via pruned actions $\beta_1, \ldots, \beta_n \in A \setminus P_\sigma$, all actions $\alpha \in P_\sigma$ commute with the pruned actions $\beta_1, \ldots, \beta_n$ and at least one action $\alpha \in P_\sigma$ remains enabled in $\sigma_n$.

The first property ensures that the pruned actions $\beta_1, \ldots, \beta_n$ are still enabled after $\alpha$ and lead to the same state ($\sigma'_n$), i.e., the order of executing $\beta_1, \ldots, \beta_n$ and $\alpha$ is irrelevant. The second avoids that deadlocks are missed when pruning states $\sigma_1, \ldots, \sigma_n$. To compute the POR set $P_\sigma$ without computing pruned states $\sigma_1, \ldots, \sigma_n$ (which would defeat the purpose of the reduction it is trying to attain in the first place), Stubborn POR uses static analysis to ‘predict’ the future from $\sigma$, i.e., to over-estimate the $\sigma$-reachable actions $A \setminus P_\sigma$, e.g.: $\beta_1, \ldots, \beta_n$.

Lipton or transaction reduction (TR) \cite{40}, on the other hand, identifies sequential blocks in the actions $A_i$ of each thread $i$ that can be grouped into transactions. A transaction $\alpha_1 \ldots \alpha_k \ldots \alpha_n \in A_i^*$ is replaced with an atomic action $\alpha$ which is its sequential composition, i.e. $\alpha = \alpha_1 \circ \ldots \circ \alpha_k \circ \ldots \circ \alpha_n$. Consequently, any trace $\sigma_1 \xrightarrow{\alpha_1} \sigma_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k} \ldots \xrightarrow{\alpha_{n-1}} \sigma_n \xrightarrow{\alpha_n} \sigma_{n+1}$ is replaced by $\sigma_1 \xrightarrow{\alpha} \sigma_{n+1}$;
making state $\sigma_2, \ldots, \sigma_n$ internal. Thereby, internal states disallow all interleavings of other threads $j \neq i$, i.e., remote actions $A_j$ are not fired at these states. Similar to POR, this pruning can reduce reachable states. Additionally, internal states can also be discarded when irrelevant to the model checking problem.

In the database terminology of origin [42], a transaction must consist of:

- A pre-phase, containing actions $\alpha_1, \ldots, \alpha_{k-1}$ that may gather required resources,
- a single commit action $\alpha_k$ possibly interfering with remote actions, and
- a post-phase $\alpha_{k+1}, \ldots, \alpha_n$, possibly releasing resources (e.g. via unlocking them).

In the pre- and post-phase, the actions (of a thread $i$) must commute with all remote behavior, i.e. all actions $A_j$ of all other threads $j \neq i$ in the system.

**TR does not dynamically ‘predict’ the possible future remote actions, like POR does.** This makes the commutativity requirement needlessly stringent, as the following example shows: Consider *program1* consisting of two threads. All actions of one thread commute with all actions of the other because only local variables are accessed. *Fig. 1* (left) shows the POR and TR of this system.

```
program1 := if (fork()) { a = 0; b = 2; } else { x = 1; y = 2; }
program2 := a = b = x = y = 0; if (fork()) { program1; }
```

Now assume that a parallel assignment is added as initialization code yielding *program2* above. *Fig. 1* (right) shows again the reductions. Suddenly, all actions of both threads become dependent on the initialization, i.e. neither action $a = 0$; nor action $b = 2$; commute with actions of other threads, spoiling the formation of a transaction $\text{atomic}\{a = 0; b = 2;\}$ (idem for $\text{atomic}\{x = 1; y = 2;\}$). Therefore, TR does not yield any reduction anymore (not drawn). Stubborn set POR [47], however, still reduces *program2* like *program1*, because, using static analysis, it ‘sees’ that the initialization cannot be fired again.

In the current paper, we show how TR can be made dynamic in the same sense as stubborn set POR [48], so that the previous example again yields the maximal reduction. Our work is based on the prequel [26], where we instrument programs in order to obtain dynamic TR for *symbolic model checking*. While [26] premiered dynamically growing and shrinking transactions, its focus on symbolic

---

*Fig. 1: Transition systems of program1 (left) and program2 (right). Thick lines show optimal (Stubborn set) POR. Curly lines show a TR (not drawn in the right figure).*
model checking complicates a direct comparison with other dynamic techniques such as POR. The current paper therefore extends this technique to enumerative model checking, which allows us to get rid of the heuristic conditions from [26] by replacing them with the more general stubborn set POR method. While we can reduce the results in the current paper to the reduction theorem of [26], the new focus on enumerative model checking provides opportunities to tailor reductions on a per-state basis and investigate TR more thoroughly. This leads to various contributions:

1. A ‘Stubborn’ TR algorithm (STR) more dynamic/general than TR in [26].
2. An open source implementation of (stubborn) TR in the model checker LTSmin.
3. Experiments comparing TR and POR for the first time in Sec. 5.

Moreover, in Sec. 4, we show analytically that unlike stubborn POR:

1. Computing optimal stubborn TR is tractable and reduction is not heuristic.
2. Stubborn TR can exploit right-commutativity and prune (irrelevant) deadlocks (while still preserving invariants as per Th. 4).

On the other hand, stubborn POR is still more effective for checking for absence of deadlocks and reducing massively parallel systems. Various open problems, including the combination of TR and POR, leave room for improvement.

The current paper is the technical report version of [36]. Proofs of theorems and lemmas can be found in Sec. A.

2 Preliminaries

Concurrent transition systems. We assume a general process-based semantic model that can accommodate various languages. A concurrent transition system (CTS) for a finite set of processes $P$ is tuple $TS \equiv (S, T, A, A_0)$. Transitions are relations between states and actions: $T \subseteq S \times A \times S$. We write $\alpha_i$ for $\alpha \in A_i$, $\sigma \xrightarrow{\alpha_i} \sigma'$ for $\langle \sigma, \alpha_i, \sigma' \rangle \in T$, $T_i$ for $T \cap (S \times A_i \times S)$, $T_\alpha$ for $T \cap (S \times \{\alpha\} \times S)$, $\alpha \rightarrow$ for $\{\langle \sigma, \sigma' \rangle \mid \langle \sigma, \alpha, \sigma' \rangle \in T \}$, and $\rightarrow_i$ for $\{\langle \sigma, \sigma' \rangle \mid \langle \sigma, \alpha, \sigma' \rangle \in T \}$.

State space exploration can be used to show invariance of a property $\varphi$, e.g., expressing mutual exclusion, written: $R(TS) \models \varphi$. This is done by finding all reachable states $\sigma$, i.e., $R(TS) \models \{\sigma \mid \exists \sigma_0 \rightarrow^* \sigma\}$, and show that $\sigma \in \varphi$.

Notation. We let $en(\sigma)$ be the set of actions enabled at $\sigma$: $\{\alpha \mid \exists \sigma_0 \rightarrow^* \sigma\}$, and $\overline{en}(\sigma) = A \setminus en(\sigma)$. We let $R;Q$ and $RQ$ denote the sequential composition of two binary relations $R$ and $Q$, defined as: $\{x, z \mid \exists y : (x, y) \in R \land (y, z) \in Q\}$. Let $R \subseteq S \times S$ and $X \subseteq S$. Then left restriction of $R$ to $X$ is $X \parallel R \equiv R \cap (X \times X)$, and right restriction is $R \parallel X \equiv R \cap (S \times X)$. The complement of $X$ is denoted $\overline{X} \equiv S \setminus X$ (the universe of all states remains implicit in this notation). The inverse of $R$ is $R^{-1} \equiv \{\langle x, y \rangle \mid \langle y, x \rangle \in R\}$.

POR relations. Dependence is a well-known relation used in POR. Two actions $\alpha_1, \alpha_2$ are dependent if there is a state where they do not commute, hence

\footnote{Symbolic approaches can be viewed as reasoning over sets of states, and therefore cannot easily support fine-grained per-state POR/TR analyses.}
we first define commutativity. Let \( c \triangleq \{ \sigma \mid \exists (\sigma, \alpha_1, \sigma') \land (\sigma, \alpha_2, \sigma'') \in T \} \). Now:

\[
\begin{align*}
\frac{\alpha_1 \to \alpha_2}{\alpha_1 \to \alpha_2} & \triangleq c \ | \ \alpha_1 \to \alpha_2 \ = \ c \ | \ \alpha_1 \to \alpha_2 \ (\alpha_1, \alpha_2 \ \text{strongly-commute}) \\
\frac{\alpha_1 \to \alpha_2}{\alpha_1 \to \alpha_2} & \triangleq \alpha_1 \to \alpha_2 \ = \ \alpha_2 \to \alpha_1 \ (\alpha_1, \alpha_2 \ \text{commute, also } \alpha_1 \bowtie \alpha_2) \\
\frac{\alpha_1 \to \alpha_2}{\alpha_1 \to \alpha_2} & \triangleq \alpha_1 \to \alpha_2 \subseteq \alpha_2 \to \alpha_1 \ (\alpha_1 \ \text{right-commutes with } \alpha_2) \\
\frac{\alpha_1 \to \alpha_2}{\alpha_1 \to \alpha_2} & \triangleq \alpha_1 \to \alpha_2 \supseteq \alpha_2 \to \alpha_1 \ (\alpha_1 \ \text{left-commutes with } \alpha_2)
\end{align*}
\]

Left / right commutativity allows actions to be prioritized / delayed over other actions without affecting the end state. Eq. 1 illustrates this by quantifying of the states: Action \( \alpha_1 \) right-commutes with \( \alpha_2 \), and vice versa \( \alpha_2 \) left-commutes with \( \alpha_1 \). Full commutativity \((\bowtie)\) always allows both delay and prioritization for any serial execution of \( \alpha_1, \alpha_2 \), while strong commutativity only demands full commutativity when both actions are simultaneously enabled, as shown in Eq. 2 for deterministic actions \( \alpha_1 / \alpha_2 \) (Eq. 2 is only for an intuition and does not illustrate the non-deterministic case, which is covered by \( \bowtie \)). Left / right / strong dependence implies lack of left / right / strong commutativity, e.g.: \( \alpha \not\bowtie \alpha_2 \).

Note that typically: \( \forall i, \alpha, \beta \in A_1: \alpha \not\bowtie \beta \) due to e.g. a shared program counter. Also note that if \( \alpha_1 \bowtie \alpha_2 \) then \( \alpha_1 \) never enables \( \alpha_2 \), while strong commutativity implies that neither \( \alpha \) disables \( \beta \), nor vice versa.

A lock(/unlock) operation right(/left)-commutes with other locks and unlocks. Indeed, a lock never enables another lock or unlock. Neither do unlocks ever disable other unlocks or locks. In the absence of an unlock however, a lock also attains left-commutativity as it is mutually disabled by other locks. Because of the same disabling property, two locks however do not strongly commute.

Finally, a necessary enabling set (NES) of an action \( \alpha \) and a state \( \sigma_1 \) is a set of actions that must be executed for \( \alpha \) to become enabled, formally:

\[
\forall E \in \text{nes}_{\sigma_1}(\alpha), \sigma_1 \stackrel{\alpha_1, \ldots, \alpha_n}{\rightarrow} \sigma_2, E \in \text{en}(\sigma_2) \Rightarrow E \cap \{\alpha_1, \ldots, \alpha_n\} \neq \emptyset.
\]

An example of an action \( \alpha \) with two NESs \( E_1, E_2 \in \text{nes}_{\sigma}(\alpha) \) is a command guarded by \( g \) in an imperative language: When \( \alpha \in \text{en}(\sigma) \), then either its guard \( g \) does not hold in \( \sigma \), and \( E_1 \) consists of all actions enabling \( g \), or its program counter is not activated in \( \sigma \), and \( E_2 \) consists of all actions that label the edges immediately before \( \alpha \) in the CFG of the process that \( \alpha \) is part of.

**POR.** POR uses the above relations to find a subset of enabled actions \( \text{por}(\sigma) \subseteq \text{en}(\sigma) \) sufficient for preserving the property of interest. Commutativity is used to ensure that the sets \( \text{por}(\sigma) \) and \( \text{en}(\sigma) \setminus \text{por}(\sigma) \) commute, while the NES is used to ensure that this mutual commutativity holds in all future behavior. The next section explains how stubborn set POR achieves this.

POR gives rise to a CTS \( \tilde{\mathcal{T}} \equiv (\tilde{S}, \tilde{T}, A, \sigma_0), \tilde{T} \equiv \{\langle \sigma, \alpha, \sigma' \rangle \in T \mid \alpha \in \text{por}(\sigma)\} \), abbreviated \( \sigma \longrightarrow \sigma' \). It is indeed reduced, since we have \( \mathcal{R}(\tilde{\mathcal{T}}) \subseteq \mathcal{R}(\mathcal{T}) \).

**Transaction reduction.** (Static) transaction reduction was devised by Lipton \([10]\). It merges multiple sequential statements into one atomic operation,
thereby radically reducing the reachable states. An action $\alpha$ is called a right/left mover if and only if it commutes with actions from all other threads $j \neq i$:

$$\frac{\alpha}{\exists} \bigcup_{j \neq i} \frac{}{} \quad (\alpha \text{ is a right mover}) \quad \frac{\alpha}{\equiv} \bigcup_{j \neq i} \frac{}{} \quad (\alpha \text{ is a left mover})$$

Both-movers are transitions that are both left and right movers, whereas non-movers are neither. The sequential composition of two movers is also a corresponding mover, and vice versa. Moreover, one may always safely classify an action as a non-mover, although having more movers yields better reductions.

Examples of right-movers are locks, P-semaphores and synchronizing queue operations. Their counterparts; unlock, V-semaphore and enqueue ops, are left-movers. Their behavior is discussed above using locks and unlocks as an example.

Lipton reduction only preserves halting. We present Lamport’s [39] version, which preserves safety properties such as $\square \varphi$, i.e. $\varphi$ is an invariant. Any sequence $\alpha_1, \ldots, \alpha_n$ can be reduced to a single action $\alpha$ s.t. $\frac{\alpha_1}{\Rightarrow} \ldots \frac{\alpha_{k-1}}{\Rightarrow} \frac{\alpha_k}{\equiv} \ldots \frac{\alpha_n}{\Rightarrow} \frac{\alpha}{\Rightarrow} \frac{i}{\not\equiv}$ (i.e. a compound statement with the same local behavior), if for some $1 \leq k < n$:

L1 actions before the commit $\alpha_k$ are right movers: $\frac{\alpha_1}{\Rightarrow} \ldots \frac{\alpha_{k-1}}{\Rightarrow} \frac{\alpha_k}{\equiv} \ldots \frac{\alpha_n}{\Rightarrow} \frac{i}{\not\equiv}$,

L2 actions after the commit $\alpha_k$ are left movers: $\frac{\alpha_k}{\equiv} \ldots \frac{\alpha_n}{\Rightarrow} \frac{i}{\not\equiv}$,

L3 actions after $\alpha_1$ do not block: $\forall \sigma \exists \sigma' \exists \sigma \frac{\alpha_1}{\Rightarrow} \ldots \frac{\alpha_n}{\Rightarrow} \sigma', \text{ and}$

L4 $\varphi$ is not disabled by $\frac{\alpha_1}{\Rightarrow} \ldots \frac{\alpha_n}{\Rightarrow} \frac{i}{\not\equiv}$, nor enabled by $\frac{\alpha_k}{\equiv} \ldots \frac{\alpha_n}{\Rightarrow} \frac{i}{\not\equiv}$.

The example (right) shows the evolution of a trace when a reduction with $n=3$, $k=2$ is applied. Actions $\beta_1, \ldots, \beta_4$ are remote. The pre-action $\alpha_1$ is first moved towards the commit action $\alpha_2$. Then the same is done with the post-action $\alpha_3, L1$ resp. $L2$ guarantee that the trace’s end state $\sigma_8$ remains invariant, $L3$ guarantees its existence and $L4$ guarantees that $\sigma_4 \notin \varphi \Rightarrow \sigma_5 \notin \varphi$ and $\sigma_6 \notin \varphi \Rightarrow \sigma_7 \notin \varphi$ (preserving invariant violations $\neg \varphi$ in the reduced system without $\sigma_4$ and $\sigma_6$). The subsequent section provides a dynamic variant of TR.

3 Stubborn Transaction Reduction

The current section gradually introduces stubborn transaction reduction. First, we introduce a stubborn set definition that is parametrized with different commutativity relations. In order to have enough luggage to compare POR to TR in Sec. 4, we elaborate here on various aspects of stubborn POR and compare our definitions to the original stubborn set definitions. We then provide a definition for dynamic left and right movers, based on the stubborn set parametrized with left and right commutativity. Finally, we provide a definition of a transaction system, show how it is reduced and provide an algorithm to do so. This demonstrates that TR can be made dynamic in the same sense as stubborn sets are dynamic. We focus in the current paper on the preservation of invariants. But since deadlock preservation is an integral part of POR, it is addressed as well.
3.1 Parametrized stubborn sets

We use stubborn sets as they have advantages compared to other traditional POR techniques [54] Sec. 4. We first focus on a basic definition of the stubborn set that only preserves deadlocks. The following version is parametrized (with $\star$).

**Definition 1 ($\star$-stubborn sets).** Let $\star \in \{\leftarrow, \rightarrow, \leftrightarrow\}$. A set $B \subseteq A$ is $\star$-stubborn in the state $\sigma$, written $\text{st}^\star(B)$, if:

- **D0** $\text{en}(\sigma) \neq \emptyset \Rightarrow B \cap \text{en}(\sigma) \neq \emptyset$ (include an enabled action, if one exists)
- **D1** $\forall \alpha \in B \cap \text{en}(\sigma): \exists E \in \text{nes}_{\star}(\alpha): E \subseteq B$ (for disabled $\alpha$ include a NES)
- **D2** $\forall \alpha \in B \cap \text{en}(\sigma), \beta \not\leq \alpha: \beta \in B$ (for enabled $\alpha$ include $\star$-dependent actions)

Notice that a stubborn set $B$ includes actions disabled in $\sigma$ to reason over future behavior with [D1]. Actions $\alpha \in B$ commute with $\beta \in \text{en}(\sigma) \setminus B$ by [D2] but also with $\beta' \in \text{en}(\sigma')$ for $\sigma \xrightarrow{\beta} \sigma'$, since [D1] ensures that $\beta$ cannot enable any $\gamma \in B$ (ergo $\beta' \notin B$). Th. 1 formalizes this. From $B$, the reduced system is obtained by taking $\text{por}(\sigma) \triangleq \text{en}(\sigma) \cap B$: It preserves deadlocks. But not all $\star$-parametrizations lead to correct reductions w.r.t. deadlock preservation. We therefore briefly relate our definition to the original stubborn set definitions. The above definition yields three interpretations of a set $B \subseteq A$ for a state $\sigma$.

- If $\text{st}^\star(B)$, then $B$ coincides with the original strong stubborn set [47,48].
- If $\text{st}_\star(B)$, then $B$ approaches the weak stubborn set in [37], a simplified version of [49], except that it lacks a necessary key action (from [49] Def. 1.17).
- If $\text{st}_\star(B)$, then $B$ may also yield an invalid POR, as it would consider two locking operations independent and thus potentially miss a deadlock.

This indicates that POR, unlike TR, cannot benefit from right-commutativity. The consequences of this difference are further discussed in [Sec. 4]. The strong version of our bare-bone stubborn set definition, on the other hand, is equivalent to the one presented [49] and thus preserves the ‘stubbornness’ property (Th. 1). If we define semi-stubbornness, written $\text{sst}^\star$, like stubbornness minus the $\text{D0}$ requirement, then we can prove a similar theorem for semi-stubborn sets (Th. 2). This ‘stubbornness’ of semi-$\rightarrow$ and semi-$\leftarrow$ stubborn sets is used below to define dynamic movers. First, we briefly return our attention to stubborn POR, recalling how it preserves properties beyond deadlocks and the computation of $\text{st}_\sigma$.

**Theorem 1 ([49]).** If $B \subseteq A$, $\text{st}^\star(B)$ and $\sigma \xrightarrow{\beta} \sigma'$ for $\beta \notin B$, then $\text{st}^\star(B)$.

---

3 [D0] is generally not preserved with left-commutativity ($\star = \leftarrow$), as $\beta \notin B$ may disable $\alpha \in B$. Consequently, $\beta$ may lead to a deadlock. Because POR prunes all $\beta \notin B$, $\text{st}^\star(B)$ is not a valid reduction (it may prune deadlocks). The key action repairs this by demanding at least one key action $\alpha$, which strongly commutes, i.e., $\forall \beta \in B: \alpha \not\leq \beta$, which by virtue of strong commutativity cannot be disabled by any $\beta \notin B$.

4 We will show that semi-stubbornness, i.e., $\text{sst}^\star(B)$ (without key), is sufficient for stubborn TR, which may therefore prune deadlocks. Contrarily, invariant-preserving stubborn POR is strictly stronger than the basic stubborn set (see below), and hence also preserves all deadlocks. (This is relevant for the POR/TR comparison in [Sec. 4].)
Theorem 2. If $B \subseteq A$, $\text{sst}^*_\sigma(B)$ and $\sigma \xrightarrow{\beta} \sigma'$ for $\beta \notin B$, then $\text{sst}^*_\sigma(B)$ for $\star \in \{\leftarrow, \leftrightarrow\}$, as well as for $\star \in \{\rightarrow\}$ provided that $\beta$ does not disable a stubborn action, i.e., $\text{en}(\sigma) \cap B \subseteq \text{en}(\sigma') \cap B$.

Stubborn sets for safety properties. To preserve a safety property such as $\square \varphi$ (i.e. $\varphi$ is invariant), a stubborn set $B$ ($\text{sst}^*_\sigma(B) = \text{true}$) needs to satisfy two additional requirements [50] called $S$ for safety and $V$ for visibility. To express $V$, we denote actions enabling $\varphi$ with $A^\varphi$ and those disabling the proposition with $A^\varphi$. Those combined form the visible actions: $A^\varphi_{\text{vis}} \triangleq A^\varphi \setminus A^\varphi$. For $S$, recall that $\sigma \xrightarrow{\alpha} \sigma'$ is a reduced transition. Ignoring states disrespect $S$.

- $S \quad \forall \beta \in \text{en}(\sigma): \exists \sigma': \sigma \xrightarrow{\alpha \star} \sigma' \land \beta \in \text{por}(\sigma')$ (never keep ignoring pruned actions)
- $V \quad B \cap \text{en}(\sigma) \cap A^\varphi_{\text{vis}} \neq \emptyset \Rightarrow A^\varphi_{\text{vis}} \subseteq B$ (either all or no visible, enabled actions)

Computing stubborn sets and heuristics. POR is not deterministic as we may compute many different valid stubborn sets for the same state and we can even select different ignoring states to enforce the $S$ proviso (i.e. the state $\sigma'$ in the $S$ condition). A general approach to obtain good reductions is to compute a stubborn set with the fewest enabled actions, so that the $\text{por}(\sigma)$ set is the smallest and the most actions are pruned in $\sigma$. However, this does not necessarily lead to the best reductions as observed several times [50,55,58]. Nonetheless, this is the best heuristic currently available, and it generally yields good results [37].

The $\forall \exists$-recursive structure of Def. [1] indicates that establishing the smallest stubborn set is an NP-complete problem, which indeed it is [51]. Various algorithms exist to heuristically compute small stubborn sets [37,57]. Only the deletion algorithm [57] provides guarantees on the returned sets (that no strict subset of the return set is also stubborn). On the other hand, the guard-based approach [37] has been shown to deliver good reductions in reasonable time.

To implement the $S$ proviso, Valmari [49] provides an algorithm [49, Alg. 1.18] that yields the fewest possible ignoring states, runs in linear time and can even be performed on-the-fly, i.e. while generating the reduced transition system. It is based on Tarjan’s strongly connected component (SCC) algorithm [46].

The above methods are relevant for stubborn TR as STR also needs to compute ($\star$-)stubborn sets and avoid ignoring states (recall L3 from Sec. 2).

3.2 Reduced transaction systems

TR merges sequential actions into (atomic) transactions and in the process removes interleavings (at the states internal to the transaction) just like POR. We present a dynamic TR that decides to prolong transactions on a per-state basis. We use stubborn sets to identify left and right moving actions in each state. Unlike stubborn set POR, and much like ample-set POR [34], we rely on the process-based action decomposition to identify sequential parts of the system.

Recall that actions in the pre-phase should commute to the right and actions in the post-phase should commute to the left with other threads. We use the notion of stubborn sets to define dynamic left and right movers in Eq. 3 and 4 for $\langle \sigma, \alpha, \sigma' \rangle \in T_i$. Both mover definitions are based on semi-stubborn sets. Dynamic left movers are state-based requiring all outgoing local transitions to...
spaces on the phase for each thread.
The transaction system effectively partitions the state. 

Lemma 3. The dynamic left-moving property is never remotely disabled, i.e.: if \( M_i^{-}(\sigma_1) \land i \neq j \land \sigma_1 \xrightarrow{\beta} \sigma_2 \), then \( M_i^{-}(\sigma_2) \).

Lemma 2. Dynamic right-movers retain dynamic moveability after moving, i.e.: if \( M_i^{-}(\sigma_1, \alpha, \sigma_2) \land \sigma_1 \xrightarrow{\alpha} \sigma_2 \xrightarrow{\beta} \sigma_3 \) for \( i \neq j \), then \( \exists \sigma_4 \xrightarrow{\beta} \sigma_4 : M_i^{-}(\sigma_4, \alpha, \sigma_3) \).

To establish stubborn TR, Def. 2 first annotates the transition system \( ts \) with thread-local phase information, i.e. one phase variable for each thread that is only modified by that thread. Phases are denoted with \( \text{Ext} \) (for transaction external states), \( \text{Pre} \) (for states in the pre-phase) and \( \text{Post} \) (for states in the post phase). Because phases now depend on the commutativity established via dynamic movers Eq. 3 and \ref{eq:lemma2} the reduction (not included in the definition, but discussed below it) becomes dynamic. Lemma 3 follows easily as the definition does not yet enforce the reduction, but mostly ‘decorates’ the transition system.

**Definition 2 (Transaction system).** Let \( H \triangleq \{ \text{Ext}, \text{Pre}, \text{Post} \}^P \) be an array of local phases. The transaction system is \( CTS \, ts' \triangleq (S', T', A, \sigma_0') \) such that:

\[
S' \triangleq S \times H, \quad \sigma_0' \triangleq \langle \sigma_0, \text{Ext}^P \rangle
\]

\[
T_i' \triangleq \{(\langle \sigma, h \rangle, \alpha, \langle \sigma', h' \rangle) \in S' \times A \times S' \mid (\langle \sigma, \alpha, \sigma' \rangle) \in T_i, \forall j \neq i : h'_j = h_j, \}
\]

\[
h_i' = \begin{cases} \text{Pre} & \text{iff } h_i \neq \text{Post} \land M_i^{-}(\sigma, \alpha, \sigma') \land \alpha \notin A^x_0 \quad (5) \\ \text{Post} & \text{iff } M_i^{-}(\sigma') \land \text{en}(\sigma') \cap \text{A}_i \cap A^x_0 = \emptyset \quad (6) \\ \text{Ext} & \text{otherwise (or as alternative when Eq. 6 holds)} \quad (7) \end{cases}
\]

**Lemma 3.** Def. 2 preserves invariants: \( \mathcal{R}(ts) \models \Box \varphi \iff \mathcal{R}(ts') \models \Box \varphi \).

The conditions in Eq. 6 and Eq. 7 overlap on purpose, allowing us to enforce termination below. The transaction system effectively partitions the state spaces on the phase for each thread \( i \), i.e. \( \text{Ext}_i = \text{Post}_i \cup \text{Pre}_i \) with \( \text{Ext}_i \triangleq \{ \langle \sigma, h \rangle \mid h_i = \text{Ext} \} \), etc. The definition of \( T_i' \) further ensures three properties:

A. \( \text{Post} \) states do not transit to \( \text{Pre} \) states as \( h_i = \text{Post} \Rightarrow h'_i \neq \text{Pre} \) by Eq. 5.

B. Transitions ending in \( \text{Pre}_i \) are dynamic right movers not disabling \( \varphi \) by Eq. 5.

C. Transitions starting in \( \text{Post}_i \) are dynamic left movers not enabling \( \varphi \) by Eq. 6.

Thereby \( T_i' \) implements the (syntactic) constraints from Lipton’s TR (see Sec. 2) dynamically in the transition system, except for L3. Let \( \rightarrow_i' \triangleq \{ \langle q, q' \rangle \mid \langle q, \alpha, q' \rangle \in T_i' \} \). Next, Th. 3 defines the reduced transaction system (RTS), based primarily on the \( \rightarrow \) transition relation that only allows a thread \( i \) to transit when all other
Algorithm 1 Algorithm reducing a CTS to an RTS using $T'_i$ from Def. 2

1: $V_1, V_2, Q_1, Q_2: S'$
2: proc Search(TS $\triangleq \langle S, T, A, \sigma_0 \rangle$)
3: $Q_1 := \langle \{\sigma_0, Ext^T\} \rangle$
4: $V_1 := \emptyset$
5: while $Q_1 \neq \emptyset$
6: $Q_1 := Q_1 \setminus \{\langle \sigma, h \rangle\}$ for $\langle \sigma, h \rangle \in Q_1$
7: $V_1 := V_1 \cup \{\langle \sigma, h \rangle\}$
8: assert($V_1: h = Ext$)
9: for $i \in P$ do
10: Transaction(T, $\langle \sigma, h \rangle$, i)
11: assert($V_1 = \mathcal{R}(\mathcal{F}_S)$)
12: function SCCROOT($q, i$)
13: return $q$ is a root of bottom SCC $C$ s.t. $\forall C \subseteq Post_i \wedge C \subseteq V_2$

Theorem 3 (Reduced Transaction System (RTS)). We define for all $i$:

\[
\rightarrow_i \triangleq (\cup_{j \neq i} Ext_j) / \rightarrow_{\mathcal{F}}_i \quad \text{(i only transits when all j are external)}
\]

\[
\rightarrow_i \triangleq Ext_i / (\rightarrow_{\mathcal{F}}_i \mid \rightarrow_i \mid \text{skip internal states transition relation})
\]

The RTS is a CST $\mathcal{F}_S \triangleq \langle S', \{\langle q, a_i, q' \rangle \mid q \rightarrow_{\mathcal{F}}_i q'\}, A, \sigma_0 \rangle$. Now, provided that $\forall \sigma \in Post_i : \exists \sigma' \in Ext_i : \sigma \rightarrow_{\mathcal{F}}_i \sigma'$, we have $\mathcal{R}(TS') \models \Box \varphi \iff \mathcal{R}(\mathcal{F}_S) \models \Box \varphi$.

The following algorithm generates the RTS $\mathcal{F}_S$ of Th. 3 from a TS. The state space search is split into two: One main search, which only processes external states ($\bigcup_i Ext_i$), and an additional search (Transaction) which explores the transaction for a single thread $i$. Only when the transaction search encounters an external state, it is propagated back to the queue $Q_1$ of the main search, provided it is new there (not yet in $V_1$, which is checked at Line 26). The transaction search terminates early when an internal state $q$ is found to be subsumed by an external state already encountered in the outer search (see the $q \not\subseteq V_1$ check at Line 23). Subsumption is induced by the following order on phases, which is lifted to states and sets of states $X \subseteq S$: $\text{Pre} \subset \text{Post} \subset \text{Ext}$ with $a \subseteq b \iff a = b \vee a \cap b = \emptyset$. $\langle \sigma, h \rangle \subseteq \langle \sigma', h' \rangle \iff \sigma = \sigma' \wedge \forall i: h_i \subseteq h_i'$, and $q \subseteq X \iff \forall q' \in X: q \subseteq q'$ (for $q = \langle \sigma, h \rangle$).

Termination detection is implemented using Tarjan’s SCC algorithm as in [49]. We chose not to obfuscate the search with the rather intricate details of that algorithm. Instead, we assume that there is a function SCCROOT that identifies a unique root state in each bottom SCC composed solely of post-states. This state is then made external on Line 23 fulfilling the premise of Th. 3 ($\forall \sigma \in \text{Post}_i : \exists \sigma' \in \text{Ext}_i : \sigma \rightarrow_{\mathcal{F}}_i \sigma'$). Combined with Lemma 3 this yields Th. 4.

Theorem 4. Alg. 1 computes $\mathcal{R}(\mathcal{F}_S)$ s.t. $\mathcal{R}(TS) \models \Box \varphi \iff \mathcal{R}(\mathcal{F}_S) \models \Box \varphi$. 

threads are in an external state, thus eliminating interlearnings ($\rightarrow_i$ additionally skips internal states). The theorem concludes that invariants are preserved given that a termination criterium weaker than L3 is met: All Post, must reach an Ext, state. Monotonicity of dynamic movers plays a key role in its proof.
Finally, while the transaction system exponentially blows up the number of syntactic states (≠ reachable states) by adding local phase variables, the reduction completely hides this complexity as Th. 5 shows. Therefore, as soon as the reduction succeeds in removing a single state, we have by definition that $|\mathcal{R}(\mathcal{TS})| < |\mathcal{R}(\mathcal{TS})|$. Th. 5 also allows us to simplify the algorithm by storing transition system states $S$ instead of transaction system states $\mathcal{S}$ in $V_1$ and $Q_1$.

**Theorem 5.** Let $N \triangleq \cap_i \text{Ext}_i$. We have $|N| = |S|$ and $\mathcal{R}(\mathcal{TS}) \subseteq \mathcal{R}(\mathcal{TS})$.

## 4 Comparison between TR and POR

Stubborn TR (STR) is dynamic in the same sense as stubborn POR, allowing for a better comparison of the two. To this end, we discuss various example types of systems that either TR or POR excel at. As a basis, consider a completely independent system with $p$ threads of $n-1$ operations each. Its state space has $n^p$ states. TR can reduce a state space to $2^p$ states whereas POR yields $n \times p$ states. The question is however also which kinds of systems are realistic and whether the reductions can be computed precisely and efficiently.

**High parallelism vs Long sequences of local transitions.** POR has an advantage when $p \gg n$ being able to yield exponential reductions. Though e.g. thread-modular verification [11,41] may become more attractive in those cases.

Software verification often has to deal with many sequential actions benefitting STR, especially when VM languages such as LLVM are used [25].

**Non-determinism.** In the pre-phase, TR is able to individually reduce mutually non-deterministic transitions of one thread due to Eq. 5, which contrary to Eq. 6 considers individual actions of a thread. Consider the example on the right. It represents a system with nine non-deterministic steps in a loop. Assume one of them never commutes, but the others commute to the right. Stubborn TR is able to reduce all paths through the loop over only the right-movers, even if they constantly yield new states (and interleavings).

**Left and right movers.** While stubborn POR can handle left-commutativity using additional restrictions, STR can benefit from right-commutativity in the pre-phase and from left-commutativity in the post-phase. E.g., P/V-semaphores are right/left-movers (see Sec. 2).

**Deadlocks.** POR preserves all deadlocks, even when irrelevant to the property. TR does not preserve deadlocks at all, potentially allowing for better reductions preserving invariants.
Table 1: Movability of commonly used synchronization mechanisms

| Mechanism       | Movability          |
|-----------------|---------------------|
| pthread_create  | As this can be modeled with a mutex that is guarding the thread’s code and is initially set to locked, the create-call is an unlock and thus a left-mover. |
| pthread_join    | Using locking similar to create, join becomes a lock and thus a right-mover. |
| Re-entrant locks| Right / left movers |
| Wait/notify/notifyAll | Can all three be split into right and left moving parts |

The following example deadlocks because of an invalid locking order. TR can still reduce the example to four states, creating maximal transactions. On the other hand, POR must explore the deadlock.

\[1(m1);l(m2); x=1; u(m1);u(m2); \parallel 1(m2);l(m1); x=2; u(m1);u(m2);\]

**Processes.** STR retains the process-based definition from its ancestors [40], while stubborn POR can go beyond process boundaries to improve reductions and even supports process algebras [53,37]. In early attempts to solve the open problem of a process-less STR definition, we observed that inclusion of all actions in a transaction could cause the entire state space search to move to the SearchTransaction function.

**Tractability and heuristics.** The STR algorithm can fix the set of stubborn transitions to those in the same thread (see definitions of \(M^*_\alpha\)). This can be exploited in the deletion algorithm by fixing the relevant transitions (see the incomplete minimization approach [57]). If the algorithm returns a set with other transitions, then we know that no transaction reduction is possible as the returned set is subset-minimal [57, Th. 1]. The deletion algorithm runs in polynomial time (in the order of \(|A|^4\) [50]), hence also stubborn TR also does (on a per-state basis). Stubborn set POR, however, is \(NP\)-complete as it has to consider all subsets of actions. Moreover, a small stubborn set is merely a heuristic for optimal reductions [51] as discussed in Sec. 3.1.

**Known unknowns.** We did not consider other properties such as full safety, LTL and CTL. For CTL, POR can no longer reduce to non-trivial subsets because of the CTL proviso [17] (see [53] for support of non-deterministic transitions, like in stubborn TR). TR for CTL is an open problem.

While TR can split visibility in enabling (in the pre-phase) and disabling (in the post-phase), POR must consider both combined. POR moreover must compute the ignoring proviso over the entire state space while TR only needs to consider post-phases and thread-local steps.

The ignoring proviso [52,10,5] in POR tightly couples the possible reductions per state to the role the state plays in the entire reachability graph. This lack of locality adds an extra obstacle to the parallelization of the model checking procedure. Early results make compromises in the obtained reductions [4]. Recent results show that reductions do not have to be affected negatively even with high amounts of parallelism [38], however these results have not yet been achieved for distributed systems. TR reduction on the other hand, offers plenty of parallelization opportunities, as each state in the out search can be handed off to a separate process.
5 Experiments

We implemented stubborn transaction reduction (STR) of Algorithm 1 in the open source model checker LTSMin\footnote{http://fmt.cs.utwente.nl/tools/ltsmin/} using a modified deletion algorithm to establish optimal stubborn sets in polynomial time (as discussed in Sec. 4). The implementation can be found on GitHub\footnote{https://github.com/alaarman/ltsmin/commits/tr}. Unlike SPIN, LTSMin does not implement dynamic commutativity specifically for queues\footnote{[28]}, but because it splits queue actions into a separate action for each cell\footnote{[56]}, a similar result is achieved by virtue of the stubborn set condition\footnote{D1} in Sec. 3.1. This benefits both its POR and STR implementation.

We compare STR against (static) TR from Sec. 2. We also compare STR against the stubborn set POR in LTSMin, which was shown to consistently outperform SPIN’s ample set\footnote{[28]} implementation in terms of reductions, but with worse runtimes due to the more elaborate stubborn set algorithms (a factor 2–4)\footnote{[37]}. (We cannot compare with \footnote{[26]} due to the different input formats of VVT\footnote{[25]} and LTSMin.) Table 2 shows the models that we considered and their normal (unreduced) verification times in LTSMin. We took all models from\footnote{[37]} that contained an assertion. The inputs include mutual exclusion algorithms:

| Model    | States | Transitions | Time (sec) | Memory (MB) |
|----------|--------|-------------|------------|-------------|
| Peterson | 829099270 | 3788955584 | 4201.6556  | 12.5496     |
| GARP     | 48363145  | 247135869  | 88.3436    | 0.3421      |
| i-Prot.2 | 13168183  | 44202271   | 22.991592  | 0.8321      |
| i-Prot.0 | 36724142  | 13150952   | 0.3421     | 0.0321      |
| BRP      | 2812740   | 6166206    | 4.5921     | 0.2412      |
| MSQ      | 994819    | 3198531    | 4.4121     | 0.1212      |
| i-Prot.3 | 327358    | 978579     | 0.7912     | 0.0312      |
| i-Prot.4 | 78977     | 169177     | 0.1912     | 0.0112      |
| Small1   | 36970     | 163058     | 0.1412     | 0.0012      |
| X.509    | 9028      | 35999      | 0.0312     | 0.0012      |
| Small2   | 7496      | 32276      | 0.0812     | 0.0012      |
| SMCS     | 2909      | 10627      | 0.0212     | 0.0012      |

Table 3: Reduction runs of TR, Stubborn TR (STR) and Stubborn POR (SPOR). Reductions of states $|S|$ and transitions $|T|$ are given in percentages (reduced state space / original state space), runtimes in sec. and memory in MB. The lowest reductions (in number of states) and the runtimes are highlighted in bold.

| Model    | TR (LTSMin) $|S|$ | $|T|$ | Time | Memory | STR (LTSMin) $|S|$ | $|T|$ | Time | Memory | SPOR (LTSMin) $|S|$ | $|T|$ | Time | Memory |
|----------|----------------|-----|------|-------|--------|----------------|-----|------|-------|--------|----------------|-----|------|-------|--------|
| Peterson | 0.5 0.3           | 6.11 33.0 | 0.4 0.3 | 74.01 29.5 | 3.1 0.9 | 316.10 299.8 | 5.2 1.9 | 42.30 2463. |
| GARP     | 100 100           | 266.21 369.8 | 1.4 1.5 | 776.53 5.2 | 3.6 1.5 | 19.83 13.5 | 7.6 3.7 | 6.27 289.1 |
| i-Prot.2 | 2.1 2.4           | 3.46 2.2 | 2.1 2.4 | 4.87 2.2 | 20.2 11.9 | 13.32 21.7 | 26.1 17.6 | 4.33 246.9 |
| i-Prot.0 | 100 100           | 56.71 75.2 | 12.8 12.5 | 148.78 9.7 | 32.1 17.2 | 214.93 24.3 | 15.7 10.5 | 2.56 132.2 |
| Peterson | 1.3 1.0           | 0.36 0.5 | 1.3 1.0 | 0.85 0.5 | 7.3 2.7 | 4.24 2.4 | 14.7 6.8 | 0.24 28.9 |
| BRP      | 100 100           | 9.59 26.4 | 47.6 36.9 | 6.38 12.6 | 100 100 | 90.31 26.4 | 9.2 6.0 | 0.18 22.2 |
| MSQ      | 66.0 65.0         | 5.5 8.2 | 22.9 21.5 | 14.90 3.0 | 52.1 29.1 | 12.14 6.5 | 80.4 46.6 | 1.03 200.9 |
| i-Prot.3 | 8.0 7.4           | 0.19 0.2 | 8.0 7.4 | 0.24 0.2 | 20.7 10.4 | 0.94 0.6 | 27.0 16.5 | 0.06 5.8 |
| i-Prot.4 | 25.1 27.2         | 0.14 0.2 | 25.0 27.1 | 0.18 0.2 | 45.2 31.5 | 0.54 0.4 | 30.4 37.1 | 0.03 2.8 |
| Small1   | 8.9 18.0          | 0.03 n/a | 6.7 13.6 | 0.07 n/a | 31.2 17.7 | 0.18 0.1 | 48.4 45.1 | 0.01 0.9 |
| X.509    | 93.8 94.1         | 0.07 n/a | 19.3 16.7 | 0.06 n/a | 7.8 3.7 | 0.03 n/a | 67.3 34.3 | 0.01 1.1 |
| Small2   | 11.6 21.0         | 0.01 n/a | 8.7 15.8 | 0.01 n/a | 35.0 19.8 | 0.04 n/a | 48.3 43.8 | 0.01 0.4 |
| SMCS     | 100 100           | 0.05 0.1 | 26.1 19.6 | 0.09 n/a | 12.5 5.3 | 0.03 n/a | 41.1 19.6 | 0.01 0.7 |

Table 2: Models and their verification times in LTSMin. Time in sec. and memory use in MB. State/transition counts are the same in both LTSMin and SPIN.
rithms (peterson), protocol implementations (i-protocol, BRP, GARP, X509), a lockless queue (MSQ) and controllers (SMCS, SMALL1, SMALL2).

LTSMin runs with STR were configured according to the command line:

```
prom2lts-mc --por=str --timeout=3600 -n --action=assert m.spins
```

The option `--por=tr` enables the static TR instead. We also run all models in SPIN in order to compare against the ample set’s performance. SPIN runs were configured according to the following command lines:

```
cc -O3 -DNOFAIR -DREDUCE -DNOBOUNDCHECK -DNOCOLLAPSE -DSAFETY -DMEMLIM=100000 -o pan pan.c
./pan -m1000000 -c0 -n -w20
```

Table 3 shows the benchmark results. We observe that STR often surpasses POR (stubborn and ample sets) in terms of reductions. Its runtimes however are inferior to those of the ample set in SPIN. This is likely because we use the precise deletion algorithm, which decides the optimal reduction for STR: STR is the only algorithm of the four that does not use heuristics. The higher runtimes of STR are often compensated by the better reductions it obtains.

Only three models demonstrate that POR can yield better reductions (BRP, smcs and X.509). This is perhaps not surprising as these models do not have massive parallelism (see Sec. 4). It is however interesting to note that GARP contains seven threads. We attribute the good reductions of STR mostly to its ability to skip internal states. SPIN’s ample set only reduces the BRP better than LTSMin’s stubborn POR and STR. In this case, we found that LTSMin too eagerly identifies half of the actions of both models as visible.

**Validation.** Validation of TR is harder than of POR. For POR, we usually count deadlocks, as all are preserved, but TR might actually prune deadlocks and error states (while preserving the invariant as per Th. 4). We therefore tested correctness of our implementation by implementing methods that check the validity of the returned semi-stubborn sets. Additionally, we maintained counters for the length of the returned transactions and inspected the inputs to confirm validity of the longest transactions.

6 Related Work

Lipton’s reduction was refined multiple times [39,21,7,6,45]. Flanagan et al. [11,15] and Qadeer et al. [14,16,13] have most recently developed transactions and found various applications. The reduction theorem used to prove the theorems in the current paper comes from our previous work [26], which in turn is a generalized version of [13]. Our generalization allows the direct support of dynamic transactions as already demonstrated for symbolic model checking with IC3 in [26].

Despite a weaker theorem, Qadeer and Flanagan [13] can also dynamically grow transactions by doing iterative refinement over the state space exploration. This contrasts our approach, which instead allows on-the-fly adaptation of movability (within a single exploration). Moreover, [13] bases dynamic behavior on exclusive access to variables, whereas our technique can handle any kind of dependency captured by the general stubborn set POR relations.
Cartesian POR \cite{23} is a form of Lipton reduction that builds transactions during the exploration, but does not exploit left/right commutativity. The leap set method \cite{44} treats disjoint reduced sets in the same state as truly concurrent and executes them as such: The product of the different disjoint sets is executed from the state, which entails that sequences of actions are executed from the state. This is where the similarity with the TR ends, because in TR the sequences are formed by sequential actions, whereas in leap sets they consist of concurrent actions, e.g., actions from different processes. Recently, trace theory has been generalized to include ‘steps’ by Ryszard et al. \cite{29}. We believe that this work could form a basis to study leap sets and TR in more detail.

Various classical POR works were mentioned, e.g. \cite{48,20,34}. How ‘persistent sets’ \cite{20} / ‘ample sets’ \cite{34} relate to stubborn set POR is explained in \cite[Sec. 4]{54}. Sleep sets \cite{19} form an orthogonal approach, but in isolation only reduce the number of transitions. Dwyer et al. \cite{8} propose dynamic techniques for object-oriented programs. Completely dynamic approaches exist \cite{12,33}. Recently, even optimal solutions were found \cite{1,43,2}. These approaches are typically stateless however, although still succeed in pruning converging paths sometimes (e.g., \cite{43}). Others aim at making dependency more dynamic \cite{18,28,35}.

Symbolic POR can be more static for reasons discussed in Footnote \cite{2}, e.g., \cite{3}. Therefore, Grumberg et al. \cite{22} present underapproximation-widening, which iteratively refines an under-approximated encoding of the system. In their implementation, interleavings are constrained to achieve the under-approximation. Because refinement is done based on verification proofs, irrelevant interleavings will never be considered. Other relevant dynamic approaches are peephole and monotonic POR by Wang et al. \cite{59,31}. Like sleep sets \cite{20}, however, these methods only reduce the number of transitions. While a reducing transitions can speed up symbolic approaches by constraining the transition relation, it is not useful for enumerative model checking, which is strongly limited by the amount of unique states that need to be stored in memory.

Kahlon et al. \cite{30} do not implement transactions, but encode POR for symbolic model checking using SAT. The “sensitivity” to locks of their algorithm can be captured in traditional stubborn sets as well by viewing locks as normal “objects” (variables) with guards, resulting in the subsumption of the “might-be-the-first-to-interfere-modulo-lock-acquisition” relation \cite{30} by the “might-be-the-first-to-interfere” relation \cite{30}, originally from \cite{20}.

Elmas et al. \cite{9} propose dynamic reductions for type systems, where the invariant is used to weaken the mover definition. They also support both right and left movers, but do automated theorem proving instead of model checking.

7 Conclusion

We presented a more dynamic version of transaction reduction (TR) based on techniques from stubborn set POR. We analyzed several scenarios for which either of the two approaches has an advantage and also experimentally compared
both techniques. We conclude that TR is a valuable alternative to POR at least for systems with a relatively low amount of parallelism.

Both in theory and practice, TR showed advantages to POR, but vice versa as well. Most strikingly, TR is able to exploit various synchronization mechanisms in typical parallel programs because of their left and right commutativity. While not preserving deadlocks, its reductions can benefit from omitting them. These observations are supported by experiments that show better reductions than a comparably dynamic POR approach for systems with up to 7 threads. We observe that the combination POR and TR is an open problem.

References

1. Parosh Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Optimal dynamic partial order reduction. In POPL, pages 373–384. ACM, 2014.
2. Elvira Albert, Puri Arenas, María García de la Banda, Miguel Gómez-Zamalloa, and Peter J. Stuckey. Context-sensitive dynamic partial order reduction. In Computer Aided Verification, pages 526–543. Springer, 2017.
3. R. Alur et al. Partial-order reduction in symbolic state space exploration. In CAV, volume 1254 of LNCS, pages 340–351. Springer, 1997.
4. J. Barnat, L. Brim, and P. Ročkai. Parallel Partial Order Reduction with Topological Sort Proviso. In SEFM, pages 222–231. IEEE, 2010.
5. D. Bošnački and G. Holzmann. Improving spin’s partial-order reduction for breadth-first search. In SPIN, volume 3639 of LNCS, pages 91–15. Springer, 2005.
6. Ernie Cohen and Leslie Lamport. Reduction in TLA. In CONCUR, volume 1466 of LNCS, pages 317–331. Springer, 1998.
7. Thomas W. Doeppner, Jr. Parallel program correctness through refinement. In POPL, pages 155–169. ACM, 1977.
8. R. Alur et al. Exploiting object escape and locking information in partial-order reductions for concurrent object-oriented programs. FMSD, 25(2-3):199–240, 2004.
9. Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. A calculus of atomic actions. In POPL, pages 2–15. ACM, 2005.
10. S. Evangelista and C. Pajault. Solving the Ignoring Problem for Partial Order Reduction. STTT, 12:155–170, 2010.
11. Cormac Flanagan, Stephen N. Freund, and Shaz Qadeer. Thread-Modular Verification for Shared-Memory Programs, pages 262–277. Springer, 2002.
12. Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In POPL, volume 40 (1), pages 110–121. ACM, 2005.
13. Cormac Flanagan and Shaz Qadeer. Transactions for software model checking. ENTCS, 89(3):518 – 539, 2003. Software Model Checking.
14. Cormac Flanagan and Shaz Qadeer. A type and effect system for atomicity. In PLDI, pages 338–349. ACM, 2003.
15. Cormac Flanagan and Shaz Qadeer. Types for atomicity. SIGPLAN Not., 38(3):1–12, January 2003.
16. Stephen N. Freund and Shaz Qadeer. Checking concise specifications for multi-threaded software. Journal of Object Technology, 3, 2004.
17. Rob Gerth, Ruurd Kuiper, Doron Peled, and W. Penczek. A partial order approach to branching time logic model checking. In TCS, pages 130–139. IEEE, 1995.
18. P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In CAV, volume 697 of LNCS, pages 438–449. Springer, 1993.
19. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *FMSD*, 2:149–164, 1993.
20. Patrice Godefroid, editor. *Partial-Order Methods for the Verification of Concurrent Systems*, volume 1032 of *LNCS*. Springer, 1996.
21. Pascal Gribomont. Atomicity refinement and trace reduction theorems. In *CAV*, volume 1102 of *LNCS*, pages 311–322. Springer, 1996.
22. Orna Grumberg et al. Proof-guided underapproximation-widening for multiprocess systems. In *POPL*, pages 122–131. ACM, 2005.
23. Guy Gueta, Cormac Flanagan, Eran Yahav, and Mooly Sagiv. Cartesian partial-order reduction. In *SPIN*, volume 4595 of *LNCS*, pages 95–112. Springer, 2007.
24. Henning Günther, Alfons Laarman, Ana Sokolova, and G. Weissenbacher. Dynamic reductions for model checking concurrent software. *CoRR*, abs/1611.09318, 2016.
25. Henning Günther et al. Vienna Verification Tool: IC3 for parallel software. In *TACAS*, volume 9636 of *LNCS*, pages 954–957. Springer, 2016.
26. Henning Günther et al. Dynamic reductions for model checking concurrent software. In *VMCAI*, pages 246–265. Springer, 2017.
27. G.J. Holzmann. The model checker SPIN. *IEEE TSE*, 23:279–295, 1997.
28. G.J. Holzmann and D. Peled. An Improvement in Formal Verification. In *IFIP WG6.1 ICFDT VII*, pages 197–211. Chapman & Hall, Ltd., 1995.
29. Ryszard Janicki, Jetty Kleijn, Maciej Koutny, and Łukasz Mikulski. Step traces. *Acta Informatica*, 53(1):35–65, 2016.
30. Vineet Kahlon et al. Symbolic model checking of concurrent programs using partial orders and on-the-fly transactions. In *CAV*, volume 4144 of *LNCS*. Springer, 2006.
31. Vineet Kahlon et al. Monotonic partial order reduction: An optimal symbolic partial order reduction technique. In *CAV*, volume 5643 of *LNCS*. Springer, 2009.
32. Gijs Kant and al. *LTSmin: High-Performance Language-Independent Model Checking*, pages 692–707. Springer, 2015.
33. Harmen Kastenberg and Arend Rensink. Dynamic Partial Order Reduction Using Probe Sets, pages 233–247. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
34. S. Katz and D. Peled. An efficient verification method for parallel and distributed programs. In *REX Workshop*, volume 354 of *LNCS*, pages 489–507. Springer, 1988.
35. Shmuel Katz and Doron Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101(2):337–359, 1992.
36. A. Laarman. Stubborn Transaction Reduction. In *NASA Formal Methods*, volume (to be published) of *LNCS*. Springer, 2018.
37. A. W. Laarman, E. Pater, J. C. van de Pol, and H. Hansen. Guard-based partial-order reduction. *STTT*, online:1–22, 2014.
38. A.W. Laarman and A.J. Wijs. Partial-Order Reduction for Multi-core LTL Model Checking. In *HVC 2014*, volume 8855 of *LNCS*, pages 267–283. Springer, 2014.
39. Leslie Lamport and Fred B. Schneider. Pretending atomicity. Technical report, Cornell University, 1989.
40. Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Comm. of the ACM*, 18(12):717–721, 1975.
41. Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko. Thread-Modular Verification Is Cartesian Abstract Interpretation, pages 183–197. Springer, 2006.
42. Christos Papadimitriou. *The theory of database concurrency control*. Principles of computer science series. Computer Science Pr., 1986.
43. César Rodríguez et al. Unfolding-based Partial Order Reduction. In *CONCUR*, volume 42 of *LIPIcs*, pages 456–469. Leibniz-Zentrum fuer Informatik, 2015.
44. Hans Van Der Schoot and Hasan Ural. An improvement of partial-order verification. *Software Testing, Verification and Reliability*, 8(2):83–102, 1998.
45. Scott D. Stoller and Ernie Cohen. Optimistic synchronization-based state-space reduction. In TACAS, volume 2619 of LNCS, pages 489–504. Springer, 2003.
46. Robert Tarjan. Depth-first search and linear graph algorithms. SIAM journal on computing, 1(2):146–160, 1972.
47. A. Valmari. Error Detection by Reduced Reachability Graph Generation. In APN, pages 95–112, 1988.
48. A. Valmari. Eliminating Redundant Interleavings During Concurrent Program Verification. In PARLE, volume 366 of LNCS, pages 89–103. Springer, 1989.
49. A. Valmari. Stubborn Sets for Reduced State Space Generation. In ICATPN/APN’89, volume 483 of LNCS, pages 491–515. Springer, 1991.
50. A. Valmari. The State Explosion Problem. In Petri Nets ’96, volume 1491 of LNCS, pages 429–528. Springer, 1998.
51. A. Valmari and H. Hansen. Can Stubborn Sets Be Optimal? In ATPN, volume 6128 of LNCS, pages 43–62. Springer, 2010.
52. Antti Valmari. A Stubborn Attack On State Explosion. In CAV, LNCS, pages 156–165. Springer, 1991.
53. Antti Valmari. Stubborn set methods for process algebras. In DIMACS POMIV, POMIV ’96, pages 213–231, New York, NY, USA, 1997. AMS Press, Inc.
54. Antti Valmari and Henri Hansen. Stubborn set intuition explained. In Petri Nets and Software Engineering 2016, CEUR-WS, pages 213–232. CEUR, 2016.
55. Antti Valmari and Walter Vogler. Fair Testing and Stubborn Sets, pages 225–243. Springer, 2016.
56. F. van der Berg and A. Laarman. SpinS: Extending LTSmin with Promela through SpinJa. ENTCS, 296:95 – 105, 2013.
57. Kimmo Varpaaniemi. Finding small stubborn sets automatically. In ISCIS, Volume I, pages 133–142. Middle East Technical University, Ankara, Turkey, 1996.
58. Kimmo Varpaaniemi. On the Stubborn Set Method in Reduced State Space Generation. PhD thesis, Helsinki University of Technology, 1998.
59. Chao Wang, Zijiang Yang, Vineet Kahlon, and Aarti Gupta. Peephole partial order reduction. In TACAS, volume 4963 of LNCS, pages 382–396. Springer, 2008.
A Correctness Proofs

The current appendix contains the proofs for the lemmas and theorems in the paper. For clarity, lemmas and theorems are repeated with the same numbering as in the paper.

In Sec. 3 we defined different semi-stubborn sets, i.e., $sts_{\sigma}^-(B)$, $sts^+_{\sigma}(B)$ and $sts_{\sigma}^-(B)$. We first provide a proof of Th. 2.

**Theorem 2.** If $B \subseteq A$, $sts_{\sigma}^+(B)$ and $\sigma \overset{\beta}{\rightarrow} \sigma'$ for $\beta \notin B$, then $sts_{\sigma'}^+(B)$ for $\sigma \ni \{\rightarrow, \leftrightarrow\}$, as well as for $\sigma \ni \{\rightarrow\}$ provided that $\beta$ does not disable a stubborn action, i.e., $en(\sigma) \cap B \subseteq en(\sigma') \cap B$.

**Proof 1** Let $\beta, \sigma$ and $\sigma'$ be such that they satisfy the premise of the theorem and $\alpha \in B$. We distinguish two cases:

If $\alpha \in \overline{en}(\sigma)$, then let $\alpha, E$ be such that $E \in nes_{\alpha}(\alpha)$ and $E \subseteq B$. $D1$ remains valid for it in $\sigma'$, since $\beta$ cannot enable $\alpha$ because $D1$ holds in $\sigma$ and, by definition of NESs, we have that $E \in nes_{\sigma}(\alpha)$.

If $\alpha \in en(\sigma)$, then either $\alpha \in en(\sigma')$ or $\alpha \in \overline{en}(\sigma')$. In the former case, the conclusion of the theorem is satisfied trivially, as $D2$ also holds in $\sigma'$. For the latter case, i.e. $\alpha \in \overline{en}(\sigma')$, we consider each $\sigma \ni \{\leftrightarrow, \rightarrow, \leftarrow\}$ separately.

$\star \Rightarrow \leftarrow$: The proof is concluded, as the definition of strong commutativity $\overset{\leftrightarrow}{\Rightarrow}$, e.g., as the deterministic case illustrated by $[Eq. 3]$ ensures that if $\beta$ disables $\alpha$, then the conclusion is not met. (Note that this also concludes the proof of $[Th. 1]$.)

$\star \Rightarrow \rightarrow$: The proof is concluded, because the additional ‘provided’ condition that $en(\sigma) \cap B \subseteq en(\sigma') \cap B$ ensures that $\beta$ cannot disable $\alpha$.

$\star \Rightarrow \leftarrow$: From $D2$ we have $\alpha \overset{\leftrightarrow}{\rightarrow} \gamma$ for all $\gamma \notin B$. Since we also have $\gamma \overset{\rightarrow}{\rightarrow} \alpha$, no $\gamma \notin B$ ever (re-)enables $\alpha$ by definition of right commutativity, as discussed in Sec. 3. Therefore, $D1$ holds in $\sigma'$ (there must be some $E \in nes_{\sigma}(\alpha)$ such that $E \cap B = \emptyset$, hence $E \subseteq B$), yielding again the conclusion of the theorem.

These three cases conclude the proof. □

Before proving the monotonicity lemmata, we recall the definition of dynamic movers and Def. 2.

\[
M_i^-(\sigma) \equiv \exists B: \text{sts}_{\sigma}^-(B), B \cap en(\sigma) = A_i \cap en(\sigma) \quad (3)
\]

\[
M_i^-(\sigma, \alpha, \sigma') \equiv \exists B: \text{sts}_{\sigma}(B), \alpha \in B, B \cap en(\sigma') \subseteq A_i, B \cap en(\sigma) = \{\alpha\} \quad (4)
\]

**Lemma 1.** The dynamic left-moving property is never remotely disabled, i.e.: if $M_i^-(\sigma_1)$ and $i \neq j \land \sigma_i \overset{\beta}{\rightarrow} j \sigma_j$, then $M_i^-(\sigma_2)$.

**Proof 2** Assume the premise: $M_i^-(\sigma_1)$ with $i \neq j$ and $\sigma_1 \overset{\beta}{\rightarrow} j \sigma_2$. We derive the conclusion.

Let $B$ be such that $sts_{\sigma_1}^-(B)$ and $B \cap en(\sigma_1) = A_i \cap en(\sigma_1)$. As $j \neq i$, we may apply Th. 3 to find that $B$ is also a valid semi-$\rightarrow$-stubborn set in $\sigma_2$, i.e. $sts_{\sigma_2}^-(B)$. Moreover, $\beta$ cannot enable any $\gamma \in B \cap \overline{en}(\sigma_1)$ by $D2$, hence $B \cap en(\sigma_2) = A_i \cap en(\sigma_2)$. That together with the semi-$\rightarrow$-stubbornness of $B$ in $\sigma_2$, implies that $M_i^-(\sigma_2)$. □
Lemma 2. Dynamic right-movers retain dynamic moveability after moving, i.e.: if $M_i^{-} (\sigma_1, \alpha, \sigma_2) \cap \sigma_1 \xrightarrow{\alpha} \sigma_2 \xrightarrow{\beta} j \sigma_3$ for $i \neq j$, then $\exists \sigma_1 \xrightarrow{\beta} j \sigma_4 : M_i^{-} (\sigma_4, \alpha, \sigma_3)$.

Proof 3 Assume the premise: $M_i^{-} (\sigma_1, \alpha, \sigma_2)$ for $\alpha \in A_i$ and $\sigma_1 \xrightarrow{\alpha} \sigma_2 \xrightarrow{\beta} j \sigma_3$ for $j \neq i$. Let $B \subseteq A$ satisfy Eq. 4, i.e.: $\text{sst}_{\sigma_1}^{-} (B)$, $\alpha \in B$, $B \cap \text{en}(\sigma_2) \subseteq A_i$, and $B \cap \text{en}(\sigma_1) = \{\alpha\}$. We derive the conclusion, i.e.: $\exists \sigma_1 \xrightarrow{\beta} j \sigma_4$ such that $\text{sst}_{\sigma_1}^{-} (B)$, $\alpha \in B$, $B \cap \text{en}(\sigma_3) \subseteq A_i$, and $B \cap \text{en}(\sigma_4) = \{\alpha\}$.

First we show that $\exists \sigma_1 \xrightarrow{\beta} j \sigma_4$ and $B \cap \text{en}(\sigma_4) = \{\alpha\}$. As $\beta \in \text{en}(\sigma_2)$, we obtain $\beta \notin B$ (since $\beta \notin A_i$). Since therefore $\alpha$ right-commutes with $\beta$ by the contraposition of [D2], we obtain the commuting path $\sigma_1 \xrightarrow{\beta} j \sigma_4 \xrightarrow{\alpha} i \sigma_3$. Assume $\exists \gamma \in B \cap \text{en}(\sigma_3) \setminus \{\alpha\}$. Action $\beta$ must have enabled $\gamma$, otherwise $\gamma \in \text{en}(\sigma_1)$, contradicting our assumption that $B \cap \text{en}(\sigma_1) = \{\alpha\}$. Now, if $\sigma_1 \xrightarrow{\beta} j \sigma_4$ enables $\gamma$, by [D1] also $\beta \in B$, again contradicting the assumption. Therefore, we have $B \cap \text{en}(\sigma_4) = \{\alpha\}$.

Theorem 3 tells us that $B$ with $\text{sst}_{\sigma_1}^{-} (B)$ is also semi-stubborn in $\sigma_4$, i.e. $\text{sst}_{\sigma_4}^{-} (B)$ (Theorem 3’s additional condition that $\text{en}(\sigma_1) \cap B = \text{en}(\sigma_4) \cap B$ is met because $\text{en}(\sigma_1) \cap B = \text{en}(\sigma_4) \cap B = \{\alpha\}$ as shown above).

We now show that $B \cap \text{en}(\sigma_3) \subseteq A_i$ also holds. Assume $\exists \gamma \in B \cap \text{en}(\sigma_3) \setminus A_i$. Action $\beta$ must have enabled $\gamma$, otherwise $\gamma \in \text{en}(\sigma_2)$, contradicting our assumption that $B \cap \text{en}(\sigma_2) \subseteq A_i$. However, if $\sigma_2 \xrightarrow{\beta} j \sigma_3$ enables $\gamma$, by [D1] also $\beta \in B$, again contradicting our assumptions. Therefore, we have $B \cap \text{en}(\sigma_3) \subseteq A_i$.

The above shows that $M_i^{-} (\sigma_4, \alpha, \sigma_3)$.

Recalling [Def. 2], we see that proving its preservation property is easy:

Definition 2 (Transaction system). Let $H \triangleq \{\text{Ext}, \text{Pre}, \text{Post}\}^P$ be an array of local phases. The transaction system is $\text{CTS \ ts'} \triangleq \langle S', T', A, \sigma_0' \rangle$ such that:

$S' \triangleq S \times H$, \hspace{1cm} $\sigma_0' \triangleq \langle \sigma_0, \text{Ext}^{A} \rangle$

$T'_i \triangleq \{\langle \sigma, h, \alpha, \langle \sigma', h' \rangle \rangle \in S' \times A \times S' | (\sigma, \alpha, \sigma') \in T_i, \forall j \neq i : h'_i = h_j\} \cup \{\alpha \in A_i \}$

$h'_i = \begin{cases} \text{Pre} & \text{iff} \ h_i \neq \text{Post} \wedge M_i^{-} (\sigma, \alpha, \sigma') \wedge \alpha \notin A_i^\sigma \quad (5) \\ \text{Post} & \text{iff} \ M_i^{-} (\sigma') \wedge \text{en}(\sigma') \cap A_i \cap A_i^\sigma = \emptyset \quad (6) \\ \text{Ext} & \text{otherwise (or as alternative when Eq. 6 holds) (7)} \end{cases}$

Lemma 3. [Def. 2] preserves invariants: $R(\text{ts}) \models \Box \varphi \iff R(\text{ts'}) \models \Box \varphi$.

Proof 4 The definition ensures the bisimulation: $\{\langle \sigma, (\sigma, h) \rangle \} \in S \times S'$.
Definition 3 (thread bisimulation). An equivalence relation $R$ on the states of a CTS $\langle S, T, A, \sigma_0 \rangle$ is a thread bisimulation iff

$$\forall \sigma, \sigma', \sigma_1, i: R \Rightarrow \exists \sigma'_1: R$$

Standard bisimulation is an equivalence relation $R$ which satisfies the property from Def. 3 when the indexes $i$ of the transitions are removed. Hence, in a thread bisimulation, in contrast to standard bisimulation, the transitions performed by thread $i$ will be matched by transitions performed by the same thread $i$. As we only make use of thread bisimulations, we will often refer to them simply as bisimulations.

We can lift these bisimulations to sets of threads, by taking the equivalence closure, e.g. $\sim = Z$ being the transitive closure of the union of all $\sim = i$ for $i \in Z$. Note that $\sim = i \iff \sim = \{i\}$.

Definition 4 (commutativity up to bisimulation). Let $R$ be a thread bisimulation on a CTS $\langle S, T, A, \sigma_0 \rangle$. The right and left commutativity up to $R$ of the transition relation $\to_i$ with $\to_j$, notation $\to_i \triangleleft R \to_j$ and $\to_i \triangleright R \to_j$ are defined as follows.

$$\to_i \triangleleft R \to_j \iff \exists \sigma_3': R \to_i \triangleright R \to_j \iff \exists \sigma_3': R$$

Illustratively:

We write $\leftrightarrow \triangleleft R$ for $\leftrightarrow \triangleleft \sim = Z$.

Using these definitions, Th. 4 provides an axiomatization of the properties required for reducing the CTS using dynamic TR. The theorem is similar to the reduction theorem in [26], where it is explained in detail. A proof of correctness is provided in [24]. Most of the constraints in its premise mirror the constraints L1–L4 provided in Sec. 2. The commutativity condition however is weakened to allow commutativity up to bisimulation. Further conditions constrain the phases of the transaction system with respect to the newly added thread bisimulations.

---

7 The version in [24] does not include Item 8 and Item 9. To reason over invariant violations, it instead distinguishes separate error states $Err_i \subseteq Ext_i$. Using the path provided by [24] Th. 2, it is straightforward to show that if a bad state $\varphi$ is reachable in the complete system, then so is one reachable in the reduced system. See also the explanation of L4 at the end of Sec. 2.
Theorem 4 (Reduction). Let \( \langle X, T, A, \sigma_0 \rangle \) be a concurrent transition system, \( Y \subseteq X \) and \( \rightarrow_i \triangleq \{ (\sigma, \sigma') \mid (\sigma, \alpha, \sigma') \in T_i \} \) (as usual). For each thread \( i \), there exists a thread bisimulation relation \( \equiv_i \). For all \( i, j \neq i \) the following holds:

1. \( X = R_i \uplus L_i \uplus N_i \) (\( R_i, L_i, N_i \) (Pre, post and external) partition \( X \))
2. \( \rightarrow_i L_j \uplus L_j \uplus N_j \) (\( \rightarrow_i \) is invariant over partitions of \( j \))
3. \( L_i \parallel \rightarrow_i \emptyset \) (post does not locally reach pre)
4. \( \rightarrow_i \emptyset \) (\( \rightarrow_i \) ending in pre right commutes with \( \rightarrow_j \))
5. \( L_i \parallel \rightarrow_i \emptyset \) (\( \rightarrow_i \) starting from post left commutes with \( \rightarrow_j \))
6. \( \forall \sigma \in L_i : \exists \sigma' \in N_i : \sigma \rightarrow^* \sigma' \) (post phases terminate locally)
7. \( \equiv_i \subseteq L_i \uplus N_i \uplus N_i \) (\( \equiv_i \) entails \( j \)-phase-equality)
8. \( Y \parallel (\rightarrow_i \emptyset) \) (\( \rightarrow_i \) into pre does not disable \( Y \))
9. \( Y \parallel (L_i \parallel \rightarrow_i) \) (\( \rightarrow_i \) from post does not enable \( Y \))

Let \( \sim_i \triangleq \bigcup_{j \neq i} N_j \parallel \rightarrow_i \) (i only transits when all \( j \) are external).

Let \( \sim_i \triangleq N_i \parallel (\rightarrow_i \emptyset)^* \rightarrow_i \emptyset \) (skip internal states).

Let \( \sim \triangleq \bigcup_i \sim_i \) and \( N \triangleq \bigcup_i N_i \).

Now, if \( \sigma \rightarrow^* \sigma' \) with \( \sigma \in N \cap Y \) and \( \sigma' \in Y \), then \( \exists \sigma'' \in Y \) s.t. \( \sigma \sim^* \sigma'' \).

We will show that our transaction system of Def. 2 satisfies the premise of \textbf{Th. 4} (in the following \textbf{Lemma 4}). In the process, the most important aspects of the theorem, i.e. the movability up to bisimulation \( \equiv_X \) in \textbf{Item 4} and \textbf{Item 5} is explained. Notice that the in the right mover case, we have \( X = \{ j \} \), while in the left-mover cases we have \( X = \{ i, j \} \).

To see the challenge ahead, observe that a remote thread \( j \) can activate a dynamic mover of thread \( i \). We illustrate with an example that this dynamic behavior causes loss of commutativity in the transaction system (not in the underlying transition system), because of the phase information that the transaction system tracks. In the following, let \( q_x \triangleq (\sigma_x, h_x) \) and \( q'_x \triangleq (\sigma'_x, h'_x) \) for \( x \in \mathbb{N} \), so that we can easily track related states in both systems.

Let \( (\sigma_1, \alpha, \sigma_2) \in T_\alpha \) (in the transition system). We have \( (\sigma_2, \beta, \sigma_3) \in M_\beta^+ \), i.e. \( \beta \) is dynamic right moving (in \( \sigma_2 \)) and leads to \( \sigma_3 \). Because of its movability, the transaction system allows that \( q_3 \in \text{Pre}_j \) (see Eq. 6 of Def. 2) as the following figure shows. The figure shows the right move of \( \alpha \) (also a right mover) with respect to \( \beta \) (the gray part). The yields states \( q_4' \) and \( q_4 \) where \( \beta \) is executed before \( \alpha \).

\[
\begin{align*}
\text{Pre}_j \ni q_1 & \xrightarrow{\beta} q_4 \in \text{Post}_j, \\
\text{Pre}_j \ni q_2 & \xrightarrow{\beta} q_3' \in \text{Post}_j.
\end{align*}
\]

Because e.g. whatever action \( \gamma \) that does not commute with \( \beta \) became unreachable after \( \alpha \), \( \beta \) does not right-move from \( \sigma_1 \) where \( \alpha \) is not yet taken. We see therefore that \( q_4' \in \text{Post}_j \). Additionally, we have \( q_4 \in \text{Post}_j \) by Def. 2 (see \( \forall j \neq i : h'_j = h_j \)). Therefore, the moving operation does not commute in the transaction system as \( q_3 \neq q_3' \).
Our theorem accounts for the differing phases of $q_3$ and $q_3'$. (Apart from the $j$-phases, these states are indeed equivalent, i.e., $\sigma_3 = \sigma_3'$ and $\forall k \neq j: h_3[k] = h_3'[k]$.) To this end, the bisimulations abstract from the phase changes, showing that the behavior of the transaction system mimics that of the original transition system.

Bisimulations indeed arise naturally from the introduced phase flags: All transitions of a thread $i$ in the transaction system are copies from transitions in the original transition system that end in a state with a different $i$.

To see that the relation from Eq. 8 and show that it is a correct thread bisimulation for each thread $i$. Then we focus out attention to the nine items in the premise of Th. 4 and show how the transaction system fulfills these conditions. In the following, again let $q_x \triangleq (\sigma_x, h_x)$ and $q_x' \triangleq (\sigma_x', h_x')$ for $x \in \mathbb{N}$.

To see that the relation $\cong_i$ from Eq. 8 is a correct thread bisimulation for thread $i$ according to Def. 2 assume that $\langle\langle \sigma, h \rangle, \langle \sigma', h' \rangle \rangle \in \cong_i$ and $\langle\langle \sigma, h \rangle, \alpha, \langle \sigma'', h'' \rangle \rangle \in T_i'$. By definition, we have $\sigma' = \sigma$ and $\forall j \neq i: h_j = h_j' = h_j''$. Therefore, by Def. 2 we also have $\langle\langle \sigma', h' \rangle, \alpha, \langle \sigma'', h'' \rangle \rangle \in T_i'$ for some $h'' \in H$ such that $\forall j \neq i: h_j = h_j' = h_j'' = h_j'''$. Finally, by definition, $\langle\langle \sigma''', h''' \rangle, \langle \sigma''''', h''''' \rangle \rangle \in \cong_i$, concluding the proof that $\cong_i$ is a proper thread bisimulation.

Next, we consider how Def. 2 satisfies the items of the premise of Th. 4.\n
**Item 1** By definition of $H$, we have $\forall i: \text{Ext}_i \cup \text{Pre}_i \cup \text{Post}_i$.

**Item 2** $T_i'$ of the transaction system ensures that remote phases remain invariant: $\forall \langle\langle \sigma, h \rangle, \alpha, \langle \sigma', h' \rangle \rangle \in T_i': \forall j \neq i: h_j = h_j'$. Therefore, we have: $\forall i: \neg \sigma_i' \subseteq \text{Ext}_i^2 \cup \text{Pre}_i^2 \cup \text{Post}_i^2$.

**Item 3** Follows immediately from Eq. 3 in Def. 2.

**Item 4** Assume that $q_1 \xrightarrow{\alpha} q_2$ with $q_2 \in \text{Pre}_i$ and $q_2 \xrightarrow{\beta} q_3$. We show that there exists a path $q_1 \xrightarrow{\alpha} q_4 \xrightarrow{\alpha} q_3'$ with $q_3 \cong_i q_3'$, or illustratively:

\[
\text{Post}_i \ni q_1 \xrightarrow{\beta} q_2 \xrightarrow{\gamma} q_3 \ni \text{Pre}_i,
\]

We have $M_{\alpha}^\ast(q_1, \alpha, q_2)$ for $\alpha \in A_i$ by Eq. 4 and thus there is some $B$ such that $\alpha \cap (B, \alpha, q_2) \subseteq A_i$ and $B \cap \text{en}(\sigma_1) = \{\alpha\}$ by Eq. 4. We also have $\sigma_2 \xrightarrow{\beta} \sigma_3$ for $j \neq i$ (from $q_2 \xrightarrow{\beta} q_3$). As $\beta \in \text{en}(\sigma_2)$, we obtain $\beta \notin B$.
As this covers all the cases in the premise of Th. 4, we conclude that the lemma holds.

1. As Def. 2 only allows transitions ending in $\sigma$ when they start in $\text{Ext}$, or $\text{Pre}$, we have $q_3 \notin \text{Post}_i$. Again, following $j$, we also get $q_4 \notin \text{Post}_i$.

2. Lemma 2 yields $M^{-}_i(\sigma_4, \alpha, \sigma_3)$ as its premise is assumed above.

3. $q_2 \notin A^c_3$ follows from the initial assumption and Eq. 3. For the above, we can conclude that $q_3 \in \text{Pre}_i$. This demonstrates that also $q_3 \equiv_j q_3'$, completing this proof.

**Item 5** Assume that $q_2 \xrightarrow{\alpha} q_3$ with $q_3 \in \text{Post}_i$ and $q_1 \xrightarrow{\beta} q_2$. We show that there exists a path $q_1 \xrightarrow{\alpha} q_4 \xrightarrow{\beta} q_3'$ with $q_3 \equiv_{(i,j)} q_3'$, or illustratively:

\[
\begin{align*}
\text{Post}_i & \ni q_1 \xrightarrow{\beta} q_2 \ni q_3' \\
\xrightarrow{\sigma_3} & \ni q_3'
\end{align*}
\]

From **Item 3**, we obtain $q_1 \in \text{Post}_i$. From the assumption in the **Lemma 4**, we get $\exists q', q'' : q \xrightarrow{\alpha'} q''$ for some $\alpha' \in A_i$. Without loss of generality, let $q, q''$ be the first on this path, i.e., that is no $i$-transition on the path $q \rightarrow q''$. By Eq. 4, we obtain $M^{-}_i(\sigma')$ and $en(\sigma') \cap A_i \cap A^c_3 = \emptyset$. As that path from $\sigma'$ to $\sigma_2$ (via $\sigma_1$) merely contains transitions from threads $k \neq i$, we may apply **Lemma 3** repeatedly to find that $M^{-}_i(\sigma_1)$. Eq. 3 implies that there is some $B$ such that $\text{sst}^{-}_i(B)$ and $B \cap en(\sigma_1) = A_i \cap en(\sigma_1)$.

Because $\alpha \in en(\sigma_2) \cap B$, we must have $\alpha \in en(\sigma_1)$ by **Def. 2**. As also $\beta \notin B$ and $\text{sst}^{-}_i(B)$, we obtain that $\alpha \equiv \beta \in \sigma_1$. Therefore, there are $\sigma_1 \xrightarrow{\alpha} \sigma_4$ and $\sigma_4 \xrightarrow{\beta} \sigma_3$ and according to **Def. 2** also $q_1 \xrightarrow{\alpha} q_4$ and $q_4 \xrightarrow{\beta} q_3'$ with $q_3' \equiv (\sigma_3, h_3')$ for some $h_3'$. By **Item 2**, we have that $\forall k \neq i, j : h_3' = h_3'.k$. This yields the desired commutativity up to $\equiv_{(i,j)}$, completing this proof.

**Item 6** The assumption in **Lemma 4** fulfills this requirement immediately.

**Item 7** By definition this follows from Eq. 8.

**Item 8** Follows immediately from Eq. 6 in **Def. 2**.

**Item 9** Follows immediately from Eq. 6 in **Def. 2**.

As this covers all the cases in the premise of Th. 4, we conclude that the lemma holds.
**Theorem 3 (Reduced Transaction System (RTS)).** We define for all $i$:

\[ \bar{\rightarrow}_i \triangleq (\bigcup_{j \neq i} \text{Ext}_j) \parallel \rightarrow_i \]  

(i only transits when all $j$ are external)

\[ \bar{\rightarrow}_i \triangleq \text{Ext}_i \parallel (\bar{\rightarrow}_i \parallel \text{Ext}_i)^* \]  

(skip internal states transition relation)

The RTS is a CST $\bar{\rightarrow}_i \bar{\rightarrow}_i \triangleq \langle S', \{ \langle q, \alpha_i, q' \rangle \mid q \bar{\rightarrow}_i q' \}, A, \sigma'_0 \rangle$. Now, provided that $\forall \sigma \in \text{Post}_i : \exists \sigma' \in \text{Ext}_i : \sigma \bar{\rightarrow}_i \sigma'$, we have $R(\bar{\rightarrow}_i)$ $\models \Box \varphi$ $\iff$ $R(\rightarrow_i)$ $\models \Box \varphi$.

**Proof 6** Lemma 4 assumes termination $\forall \sigma \in \text{Post}_i : \exists \sigma' \in \text{Ext}_i : \sigma \rightarrow_i \sigma'$ and actuation $\forall \sigma \in \text{Post}_i : \exists \sigma', \sigma'' : \sigma \rightarrow_i \sigma'' \rightarrow_i \sigma$ of post phases. The termination assumption is met by the 'provided' assumption of the theorem. The actuation is met by the theorem's use of $\rightarrow_i$, which only considers the subsystem of full transactions starting with and ending in external states: $\bar{\rightarrow}_i \triangleq \text{Ext}_i \parallel (\bar{\rightarrow}_i \parallel \text{Ext}_i)^* \parallel \text{Ext}_i$. As the premise of Lemma 4 is therefore met, it follows that we can apply [Th. 4]. Therefore, if $\sigma \rightarrow_i^* \sigma'$ with $\sigma \in \bigcap_i \text{Ext}_i \cap \varphi$ and $\sigma' \in \bar{\varphi}$, then $\exists \sigma'' \in \bar{\varphi}$ s.t. $\sigma \rightarrow_i^* \sigma''$. As therefore invariant violations are preserved by the reduction, we have $R(\bar{\rightarrow}_i)$ $\not\models \Box \varphi$ $\Rightarrow$ $R(\bar{\rightarrow}_i)$ $\not\models \Box \varphi$. Because $\rightarrow_i \subseteq \bar{\rightarrow}_i$ and $\bar{\rightarrow}_i \subseteq \bar{\rightarrow}_i$, we also have the opposite $R(\bar{\rightarrow}_i)$ $\not\models \Box \varphi$ $\Rightarrow$ $R(\bar{\rightarrow}_i)$ $\not\models \Box \varphi$, i.e., as by definition the reduced system must contain a subset of the invariant violations in the full system. Taken together (conjoining the contrapositives), Th. 3 is satisfied.

**Theorem 4.** Alg. 1 computes $R(\bar{\rightarrow}_i)$ s.t. $R(\bar{\rightarrow}_i)$ $\models \Box \varphi$ $\iff$ $R(\rightarrow_i)$ $\models \Box \varphi$.

**Proof 7** The algorithm uses the approach described by Valmari [49] to ensure that $\forall \sigma \in \text{Post}_i : \exists \sigma' \in \text{Ext}_i : \sigma \rightarrow_i \sigma'$. It follows therefore that the premise of Lemma 4 holds (including the “provided that” part). From Lemma 5, we also have $R(\bar{\rightarrow}_i)$ $\models \Box \varphi$ $\iff$ $R(\rightarrow_i)$ $\models \Box \varphi$, hence: $R(\bar{\rightarrow}_i)$ $\models \Box \varphi$ $\iff$ $R(\rightarrow_i)$ $\models \Box \varphi$.

**Theorem 5.** Let $N \triangleq \bigcap_i \text{Ext}_i$. We have $|N| = |S|$ and $R(\bar{\rightarrow}_i) \subseteq R(\rightarrow_i)$.

**Proof 8** By definition of the initial state $\sigma'_0$ in Def. 2, the reduced transition relations $\leftrightarrow$, $\rightarrow_i$ in Th. 3 and basic induction.