Tighter Reachability Criteria for Deadlock-Freedom Analysis

Pedro Antonino, Thomas Gibson-Robinson, and A.W. Roscoe

Department of Computer Science, University of Oxford, Oxford, UK
{pedro.antonino,thomas.gibson-robinson,bill.roscoe}@cs.ox.ac.uk

Abstract. We combine a prior incomplete deadlock-freedom-checking approach with two new reachability techniques to create a more precise deadlock-freedom-checking framework for concurrent systems. The reachability techniques that we propose are based on the analysis of individual components of the system; we use static analysis to summarise the behaviour that might lead components to this system state, and we analyse this summary to assess whether components can cooperate to reach a given system state. We implement this new framework on a tool called DeadlOx. This implementation encodes the proposed deadlock-freedom analysis as a satisfiability problem that is later checker by a SAT solver. We demonstrate by a series of practical experiments that this tool is more accurate than (and as efficient as) similar incomplete techniques for deadlock-freedom analysis.

1 Introduction

Deadlock-checking techniques seek to establish whether a finite-state concurrent system can reach a blocked state. Complete approaches construct and search a system's state space for blocked states, and thus, they either show that a system is deadlock free or they find a deadlock, namely, a *snapshot* of the system that is both reachable and blocked. A snapshot is a tuple containing a component state per component of the concurrent system, i.e. a possible state of the system. These techniques, however, tend not to be scalable: deadlock-freedom checking quickly becomes intractable as systems grow in size.

To cope with this lack of scalability, a number of incomplete deadlock-freedom-checking techniques have been proposed [12, 6, 13, 2, 3, 5, 16]. These techniques imprecisely characterise a deadlock using *local analysis*, that is, they analyse only small parts of the system (for instance, individual components or pairs of them) to establish, conservatively, whether a system can deadlock. This imprecise characterisation makes these techniques scalable at the expense of making them incomplete, namely, they either guarantee deadlock freedom or are inconclusive. In the latter case, the system might deadlock or not.

In [2], we presented an incomplete deadlock-checking technique that significantly improves on previous frameworks that use local analysis. It attempts to use purely local analysis to show that no blocked snapshot is reachable. While this works well for many classes of systems, it does not work in cases where the

interactions of the system maintain some global invariant that prevents deadlocks too subtle to identify with our original methods. This inability is a consequence of characterising snapshots reachability using pure local analysis.

In this paper, we propose two complementary reachability criteria, based on two common sorts of global invariant, that are combined with the pure-local-analysis technique in [2] to create a more precise deadlock-freedom technique. This new deadlock-freedom technique is implemented in the DeadlOx tool, which makes use of SAT checkers and FDR3's capabilities [9]. As in [2], using the capabilities of SAT checkers means we can be ambitious in the properties of snapshots that we seek to establish.

Outline. Section 2 briefly introduces CSP's operational semantics, which is the formalism upon which our strategy is based. However, this paper can be understood purely in terms of communicating systems of LTSs, and knowledge of CSP is not a prerequisite. Section 3 presents some related incomplete deadlock-freedom-checking techniques. In Section 4, we introduce our reachability criteria. Section 5 presents our new framework for imprecise deadlock-freedom checking. Section 6 presents an experiment conducted to assess the accuracy and efficiency of our DeadlOx tool. Finally, in Section 7, we present our concluding remarks.

2 Background

Communicating Sequential Processes (CSP) [11, 19] is a notation used to model concurrent systems where processes interact, exchanging messages. Here we describe some structures used by the refinement checker FDR3 [9] in implementing CSP's operational semantics. As this paper does not depend on the details of CSP, we do not describe the details of the language or its semantics. These can be found in [19].

CSP's operational semantics interpret language terms as a *labelled transition system* (LTS).

Definition 1. *A labelled transition system is a 4-tuple* $(S, \Sigma, \Delta, \hat{s})$ *where* S *is a set of states,* Σ *is the alphabet,* $\Delta \subseteq S \times \Sigma \times S$ *is a transition relation, and* $\hat{s} \in S$ *is the starting state.*

FDR3 represents concurrent systems as *supercombinator machines.* A supercombinator machine consists of a set of component LTSs along with a set of rules that describe how components transitions should be combined. We restrict FDR3's usual definition to systems with pairwise communication, as per [13, 2].

Definition 2. *A supercombinator machine is a pair* $(\mathcal{L}, \mathcal{R})$ *where:*

- $\mathcal{L} = \langle L_1, \ldots, L_n \rangle$ *is a sequence of component LTSs;*
- $\mathcal{R}$ *is a set of rules of the form* (i, e, a) *where:*
  - $i \in \mathbb{N}$ *is a unique identifier for the rule;*
  - $e \in (\Sigma \cup \{-\})^n$ *specifies the event that each component must perform, where* $-$ *indicates that the component performs no event;* e *must also be triple-disjoint, that is, at most two components must be involved in a rule.*

  * $triple_disjoint(e) \mathrel{\widehat{=}} \forall\, i, j, k \in \{1 \ldots n\} \mid i \neq j \wedge j \neq k \wedge i \neq k \bullet$
    $e_i = - \vee e_j = - \vee e_k = -$
- *$a \in \Sigma$ is the event the machine performs.*

The *participants* of a rule are the components required to perform an event. Given a supercombinator machine, a corresponding LTS can be constructed.

Definition 3. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine where $L_i = (S_i, \Sigma_i, \Delta_i, \hat{s}_i)$. The LTS induced by $\mathcal{S}$ is the tuple $(S, \Sigma, \Delta, \hat{s})$ such that:*

- $S = S_1 \times \ldots \times S_n$;
- $\Sigma = \{i \mid \exists (i, e, a) \in \mathcal{R}\}$;
- $\Delta = \{((s_1, \ldots, s_n), j, (s'_1, \ldots, s'_n)) \mid \exists (j, (e_1, \ldots, e_n), a) \in \mathcal{R} \bullet \forall\, i \in \{1 \ldots n\} \bullet$
  $(e_i = - \wedge s_i = s'_i) \vee (e_i \neq - \wedge (s_i, e_i, s'_i) \in \Delta_i)\}$;
- $\hat{s} = (\hat{s}_1, \ldots, \hat{s}_n)$.

We slightly change the common definition of an induced LTS to focus on rule occurrences instead of system-event performances. Usually, a rule application is seen as an synchronisation between components that results in a system event. However, for our analyses, we are interested in the identifier of the rule used rather than the system event it produces.

We write $s \xrightarrow{r} s'$ if $(s, r, s') \in \Delta$. There is a path from s to s' with the sequence of rule identifiers $\langle r_1, \ldots, r_n \rangle \in \Sigma^*$, represented by $s \xrightarrow{\langle r_1, \ldots, r_n \rangle} s'$, if there exist $s_0, \ldots, s_n$ such that $s_0 \xrightarrow{r_1} s_1 \ldots s_{n-1} \xrightarrow{r_n} s_n$, $s_0 = s$ and $s_n = s'$. A trace is a path starting from the initial state. For our analyses, we will be mainly interested in the rule-identifier traces of induced LTSs.

Definition 4. *A LTS $(S, \Sigma, \Delta, \hat{s})$ deadlocks in a snapshot s if and only if the predicate $deadlocked(s)$ holds, where:*

- $deadlocked(s) \mathrel{\widehat{=}} reachable(s) \wedge blocked(s)$
- $reachable(s) \mathrel{\widehat{=}} \exists\, tr \in \Sigma^* \bullet \hat{s} \xrightarrow{tr} s$
- $blocked(s) \mathrel{\widehat{=}} \neg \exists\, s' \in S\, ;\, r \in \Sigma \bullet s \xrightarrow{r} s'$

3 Related Work

The *SDD* (State Dependency Digraph), developed by Martin in [13], is the basis of an incomplete technique that attempts to prove deadlock-freedom for triple-disjoint systems. It uses local analysis to construct the dependency digraph of a system. This framework relies on the fact that every deadlock produces a cycle in the system's dependency digraph. So, a cycle-free dependency digraph shows that a system is deadlock free. This characterisation can be efficiently checked by algorithms that detect cycles in a digraph. However, this cycle-of-dependencies characterisation for a deadlock can be rather imprecise.

In [2], we proposed *Pair*, an improved incomplete technique that checks deadlock-freedom for triple-disjoint systems. As per [13], it characterises a deadlock by analysing how pairs of components interact.

Definition 5. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine. The* pairwise projection *$\mathcal{S}_{i,j}$ of the machine $\mathcal{S}$ on components i and j is given by:*

$$\mathcal{S}_{i,j} = (\langle L_i, L_j \rangle, \{(k, (e_i, e_j), a) \mid \exists (k, (e_1, \ldots, e_n), a) \in \mathcal{R} \bullet (e_i \neq - \vee e_j \neq -)\})$$

Instead of looking for cycles of dependencies, Pair characterises a deadlock as a snapshot of the system that is fully consistent with local reachability and blocking information. We call it a *Pair candidate*. As we use local analysis to its full extent, we end up with a framework that is strictly better than the SDD.

Definition 6. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine, and $(S, \Sigma, \Delta, \hat{s})$ its induced LTS. A state $s = (s_1, \ldots, s_n) \in S$ is a* Pair candidate *iff $pair_candidate(s)$ holds, where:*

- $pair_candidate(s) \mathrel{\widehat{=}} pairwise_reachable(s) \wedge blocked(s)$
- $pairwise_reachable(s) \mathrel{\widehat{=}} \forall\, i, j \in \{1 \ldots n\} \mid i \neq j \bullet reachable_{i,j}((s_i, s_j))$

$reachable_{i,j}$ is the reachable predicate for the pairwise projection $\mathcal{S}_{i,j}$.

The analysis of pairs of components can be used to exactly characterise whether a snapshot is blocked; Pair does that. The reachability of a snapshot, however, cannot be exactly captured by this sort of local analysis. Thus, despite using pairwise-analysis to its full extent, Pair can only conservatively approximates reachability with the predicate *$pairwise_reachable(s)$*. This limitation makes such techniques unable to, in particular, show that a snapshot is unreachable if that is due to some global property of the system's behaviour. For example:

Running example 1 (From [19]). Let $\mathcal{S} = (\langle L_0, L_1, L_2 \rangle, \mathcal{R})$ be the supercombinator machine with L_0, L_1 and L_2 defined in Figure 1 and $\mathcal{R}$ the set of rules that require components to synchronise on shared events; for instance, for event $ring_1$, we have rule $(n, (ring_1, ring_1, -), ring_1)$ where n can be any unique identifier. For the sake of presentation, we use the name of an event to refer to the rule that requires its synchronisation. As τ is not synchronised, there are three rules τ_0, τ_1, τ_2, such that τ_i allows component i to perform a τ. Components can receive messages either from another component, via event $ring_i$, or from its user, via event in_i. If it holds a message, it can pass the message along, via event $ring_{i \oplus 1}$, or output the message to its user, via out_i. The τ transitions represent an internal (non-deterministic) decision of the component. The Pair candidate (s_6, s_6, s_6) is not a deadlock; this snapshot is unreachable and yet pairwise reachable. □

Running example 2. Let $\mathcal{S} = (\langle L_0, L_1, L_2 \rangle, \mathcal{R})$ be the supercombinator machine with L_0, L_1 and L_2 defined in Figure 2 and $\mathcal{R}$ the set of rules that require components to synchronise on shared events. For the sake of presentation, we use the name of an event to identify the rule requiring its synchronisation. This system implements a token ring where process L_0 has the token initially and the events tk_i represents the passage of a token from $L_{i \ominus 1}$ to L_i, where $\ominus$ is subtraction modulo 3. The Pair candidate (s_1, s_2, s_2) is not a deadlock; this snapshot is pairwise reachable but it is not reachable. □

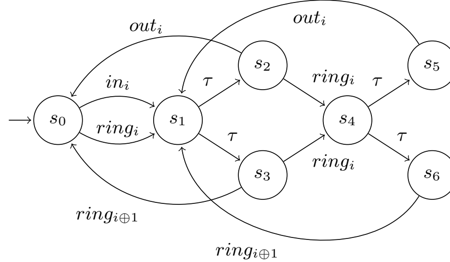

Fig. 1. LTS of component L_i where $\oplus$ represents addition modulo 3.

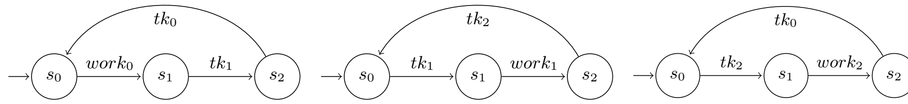

Fig. 2. LTSs of components L_0, L_1, and L_2, respectively.

To cope with this pure-local-analysis inadequacy, Martin proposed two extensions of the SDD: the *CSDD* (Coloured State Dependency Digraph) and *FSDD* (Flashing State Dependency Digraph)[13]. These extend the SDD by adding extra reachability information to a dependency, which in turn, leads to more precise cycle-of-dependencies characterisations for a deadlock. They can, in particular, prove that the previous two examples are deadlock free. As for the SDD, the characterisations proposed by these frameworks discard some local-analysis information, which could be used to increase precision, so they obtain efficiency.

4 Imprecise Reachability using Local Static Analysis

In this section, we propose two techniques to decide whether a snapshot is reachable. The techniques make use of two global invariants of our concurrent systems: to reach a snapshot, components have to agree on the order in which they synchronise on rules, and they must agree on the number of times they perform shared rules. Informally, our techniques try to show that, for a given snapshot, components cannot satisfy these invariants, so the snapshot must be unreachable. If, however, components can meet these invariants, they might be able to cooperate to reach the snapshot, and so, we conservatively assume that the snapshot is reachable. The use of these global invariants make these techniques able to prove unreachability for snapshots that are beyond the capabilities of techniques using only pure local analysis.

To check whether these global invariants are met, both techniques analyse a *component projection* that depicts the component's behaviour in terms of the system rules in which it participates rather than its own local events.

Definition 7. *Let* $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ *be a supercombinator machine, where* $L_i = (S_i, \Sigma_i, \Delta_i, \hat{s}_i)$*, and* $(S, \Sigma, \Delta, \hat{s})$ *its induced LTS. The projection of* $\mathcal{S}$ *over component* i *is given by following supercombinator machine:*

$$\mathcal{S}_i = (\langle L_i \rangle, \{(j, (e_i), a) \mid \exists (j, (e_1, \ldots, e_n), a) \in \mathcal{R} \bullet e_i \neq -\})$$

4.1 Ordering of Rules Occurrences Consistency

In the first technique, we try to show that a snapshot is unreachable by showing that components cannot agree on the order in which they cooperate to reach this snapshot. We present our technique with the help of Running Example 1.

First of all, we analyse the traces that lead each component projection to its corresponding state in the snapshot. Note that there might be infinitely many traces leading such a projection to one of its states; this happens, for instance, if there exists a trace reaching the target state that passes by a loop in the component projection's induced LTS. We summarise this set of traces with a suffix that is common to all such traces. We adapt the general framework for static analysis presented in [15] to systematically calculate $SF_{i,j}$: the longest common suffix for the traces leading component i's projection to state s_j. We call $SF_{i,j}$ an *invariant suffix* of state s_j of component i.

Definition 8. *Let* $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ *be a supercombinator machine,* $(S, \Sigma, \Delta, \hat{s})$ *its induced LTS,* $\mathcal{S}_i$ *the projection of* $\mathcal{S}$ *over component* i*, and* $L_i = (\{s_0, \ldots, s_{\mathbf{m}}\}, \Sigma_i, \Delta_i, s_0)$ *its induced LTS. When applied to* L_i*, the following static analysis framework computes a collection* $\mathbf{SF_i}$ *with* $\mathbf{m}$ *elements, where* $SF_{i,j} \in (\Sigma^* \cup \{\bot\})$ *(*$\mathbf{SF_i}$*'s* j*-th element) is a sequence of rule identifiers that we call an* invariant suffix of state s_j of component i.

- $Init = \langle\rangle$
- $D = (\{\bot\} \cup \Sigma^{\mathbf{m}}, \sqsubseteq)$*, where* $a \sqsubseteq b$ *holds if* b *is a suffix of* a *and* $\bot$ *is the least element.*
- $F_r(\bot) \mathrel{\widehat{=}} \bot$ *and* $F_r(d) \mathrel{\widehat{=}} d\,\hat{}\,\langle r \rangle$*.*

Given these three elements and $\sqcup$*, the join operator induced by the lattice* D*, the collection* $\mathbf{SF_i}$ *is the least fixed point for the following set of equations:*

- $SF_{i,0} = Init \sqcup SF_{i,0}$
- $SF_{i,j'} = F_r(SF_{i,j}) \sqcup SF_{i,j'}$*, for each* $(s_j, r, s_{j'}) \in \Delta_i$

To see how these component suffixes translate to the participation of components on the system's behaviour, we can derive an *occurrence suffix* from them. An occurrence suffix translates a sequence of rule identifiers to a sequence of global (or system-wide) rule occurrences; i.e. they represent synchronisations a component must engage on to reach the associated state.

Definition 9. *Let* $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ *be a supercombinator machine, and* $(S, \Sigma, \Delta, \hat{s})$ *its induced LTS. An occurrence variable* O^i_r *denotes the* i*-th most recent occurrence of rule* r*, and* $Rct(\bot) = \bot$ *or* $Rct(SF)$*, where* $SF \in \Sigma^*$*, gives the sequence of occurrence variables that is obtained by replacing the* i*-th most recent occurrence of rule* r *in* SF *by* O^i_r*. We use* $SFO_{i,j}$ *to denote* $Rct(SF_{i,j})$*.*

Running example 1. For the component states in the pair candidate analysed, we have the following invariant suffixes and occurrence suffixes: $SF_{0,6} = \langle \tau_0, ring_0, \tau_0 \rangle$, $SF_{1,6} = \langle \tau_1, ring_1, \tau_1 \rangle$, $SF_{2,6} = \langle \tau_2, ring_2, \tau_2 \rangle$, $SFO_{0,6} = \langle O^1_{\tau_0}, O^0_{ring_0}, O^0_{\tau_0} \rangle$, $SFO_{1,6} = \langle O^1_{\tau_1}, O^0_{ring_1}, O^0_{\tau_1} \rangle$, and $SFO_{2,6} = \langle O^1_{\tau_2}, O^0_{ring_2}, O^0_{\tau_2} \rangle$. □

Next, we present a predicate that formalises our technique. Roughly speaking, we use the clock variables clk^i_r, where clk^i_r marks the instant at which the occurrence O^i_r happened, to find a system synchronisation ordering that respects the occurrence suffixes of component states in the snapshot under analysis.

Definition 10. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine, $(S, \Sigma, \Delta, \hat{s})$ its induced LTS, and $clk(O^i_r) \mathrel{\widehat{=}} clk^i_r$. For $s = (s_{j(1)}, \ldots, s_{j(n)}) \in S$ and $occurs = \{O^a_b, \ldots, O^y_z\}$:*

$$reachable_{\mathcal{S}}(s) \mathrel{\widehat{=}} \exists\, clk^a_b, \ldots, clk^y_z \in \mathbb{N} \bullet \bigwedge_{i \in \{1 \ldots n\}} HBC(i, j(i))$$

where:

- $HBC(i, j(i)) \mathrel{\widehat{=}} \begin{cases} False & \text{if } SFO_{i,j(i)} = \bot \\ True & \text{if } SFO_{i,j(i)} = \langle\rangle \\ TC(i, j(i)) \wedge BC(i, j(i)) & \text{otherwise} \end{cases}$
- $TC(i, j(i)) \mathrel{\widehat{=}} \bigwedge_{(O,O') \in adj(SFO_{i,j(i)})} clk(O) < clk(O')$
  - ⋆ *The trace constraint (TC) enforces that a system synchronisation respects the order in which rule occurrences appear in $SFO_{i,j(i)}$.*
- $BC(i, j(i)) \mathrel{\widehat{=}} \bigwedge_{O \in dif_i(SFO_{i,j(i)})} clk(O) < clk(head(SFO_{i,j(i)}))$
  - ⋆ *A rule occurrence that requires the participation of component i but is not in $SFO_{i,j(i)}$ must have happened before the occurrences in $SFO_{i,j(i)}$; the before constraint (BC) enforces that a system synchronisation respects this principle.*
- $occurs \mathrel{\widehat{=}} \bigcup \{ \boldsymbol{SET}(SFO_{i,j(i)}) \mid i \in \{1 \ldots n\} \}$
  - ⋆ *This represents the universal set of occurrences for the component states in the snapshot under analysis.*
- $adj(SFO) \mathrel{\widehat{=}} \{(O, O') \mid \langle O, O' \rangle$ ***is a subsequence of*** $SFO\}$
  - ⋆ *This set contains the pairs of adjacent elements in the sequence SFO, where the elements in these pairs are ordered by their order in SFO.*
- $dif_i(SFO) \mathrel{\widehat{=}} \{O^l_r \mid O^l_r \in occurs \wedge i \in pts(r) \wedge O^l_r \notin \boldsymbol{SET}(SFO)\}$;
  - ⋆ *This set contains the occurrences of rules that component i participates in but are not present on SFO.*
- $pts(r) \mathrel{\widehat{=}} \{i \mid i \in \{1 \ldots n\} \wedge \exists (r, e, a) \in \mathcal{R} \bullet e_i \neq -\}$*, gives the participants of rule r.*

If this predicate holds, these HBCs (Happen-Before Constraints) are consistent and components can agree on an ordering in which they participate on these occurrences. Hence, the snapshot might be reachable. On the other hand, if the predicate is false, these constraints are inconsistent: either a component state is trivially unreachable within its own projection (for which $SF_{i,j} = \bot$), or there is an inconsistency between components happens-before orderings. Either way, components are unable to cooperate to reach the snapshot.

Running example 1. For this example's pair candidate, we get the following happens-before constraints:

1. $HBC(0,6) = clk^1_{\tau_0} < clk^0_{ring_0} \wedge clk^0_{ring_0} < clk^0_{\tau_0} \wedge clk^0_{ring_1} < clk^1_{\tau_0}$;
2. $HBC(1,6) = clk^1_{\tau_1} < clk^0_{ring_1} \wedge clk^0_{ring_1} < clk^0_{\tau_1} \wedge clk^0_{ring_2} < clk^1_{\tau_1}$;
3. $HBC(2,6) = clk^1_{\tau_2} < clk^0_{ring_2} \wedge clk^0_{ring_2} < clk^0_{\tau_2} \wedge clk^0_{ring_0} < clk^1_{\tau_2}$.

From 1, 2 and 3, we can deduce that $clk^0_{ring_0} < clk^0_{ring_2} < clk^0_{ring_1} < clk^0_{ring_0}$, this contradiction shows that $reachable((s_6, s_6, s_6))$ is false and that components cannot agree on the order in which they participate on these rule occurrences. Note that this predicate could show the pair candidate unreachable for any such system with 3 or more components. □

Given that components must synchronise on shared rules to reach snapshots, for any reachable snapshot, components must be able to, in particular, agree on the occurrences suffixes leading to this snapshot. So[1]:

Theorem 1. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine with $(S, \Sigma, \Delta, \hat{s})$ its induced LTS. For a snapshot $s \in S$, $reachable(s) \Rightarrow reachable_S(s)$.*

Hence, this predicate over-approximates reachability, and as a consequence, it can be soundly used for deadlock-freedom analysis.

4.2 Number of Rules Occurrences Consistency

In the second technique, we try to show that a snapshot is unreachable by showing that components cannot agree on the number of times they need to cooperate to reach the snapshot. We use Running Example 2 to introduce this technique.

In this technique, we summarise the traces leading component i's projection to its state s_j by an *invariant relation* $\oplus^{k,l}_{i,j}$ that relates the number of times that rules k and l have been applied in any of these traces. We can systematically calculate such a relation as follows.

Firstly, we use static analysis to compute $DS^{k,l}_{i,j}$: a set of integers in which the difference $t \downarrow k - t \downarrow l$ lies for all traces t leading component i's projection to its state s_j ($t \downarrow l$ counts the number of times rule l occurred in the trace t).

Definition 11. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine, $(S, \Sigma, \Delta, \hat{s})$ its induced LTS, $\mathcal{S}_i$ the projection of $\mathcal{S}$ over component i, and $L_i = (\{s_0, \ldots, s_{\mathbf{m}}\}, \Sigma_i, \Delta_i, s_0)$ its induced LTS. When applied to L_i and parametrised by rules k and l, the following static analysis framework computes a collection $\mathbf{DS^{k,l}_i}$ with $\mathbf{m}$ elements, where $DS^{k,l}_{i,j} \in (\{\emptyset, \mathbb{Z}\} \cup \{\{a\} \mid a \in \mathbb{Z}\})$ ($\mathbf{DS^{k,l}_i}$'s j-th element) is a set of integers called an* invariant difference set *for rules k and l and state s_j of component i.*

- *$Init = \{0\}$;*

[1] Formal proofs for all theorems in this work can be found in [4].

- $D = (\{\emptyset, \mathbb{Z}\} \cup \{\{a\} \mid a \in \mathbb{Z}\}, \subseteq)$ *the flat integer domain where* $\subseteq$ *is the usual order on sets;*
- $F_r(\{d\}) \mathrel{\widehat{=}} \begin{cases} \{d+1\} & \textbf{\textit{if}}\ r = k \\ \{d-1\} & \textbf{\textit{if}}\ r = l \\ \{d\} & otherwise \end{cases}$, $F_r(\emptyset) \mathrel{\widehat{=}} \emptyset$ *and* $F_r(\mathbb{Z}) \mathrel{\widehat{=}} \mathbb{Z}$.

Given these three elements and $\sqcup$*, the join operator induced by the lattice* D*, the collection* $\mathbf{DS_i^{k,l}}$ *is the least fixed point for the following set of equations:*

- $DS_{i,0}^{k,l} = Init \sqcup DS_{i,0}^{k,l}$
- $DS_{i,j'}^{k,l} = F_r(DS_{i,j}^{k,l}) \sqcup DS_{i,j'}^{k,l}$, *for each* $(s_j, r, s_{j'}) \in \Delta_i$

From this difference set, we can obtain $\oplus_{i,j}^{k,l}$ as follows.

Definition 12. *We define* $\oplus_{i,j}^{k,l} = Rel(DS_{i,j}^{k,l})$*, where* $Rel(DS)$ *for* $DS \in (\{\emptyset, \mathbb{Z}\} \cup \{\{d\} \mid d \in \mathbb{Z}\})$ *is:*

- $<$ *if* $DS \subseteq \{d \mid d < 0\} \wedge DS \neq \emptyset$,
- $>$ *if* $DS \subseteq \{d \mid d > 0\} \wedge DS \neq \emptyset$,
- $=$ *if* $DS = \{0\}$,
- $\bot$ *if* $DS = \emptyset$,
- $\top$ *if* $DS = \mathbb{Z}$;

and $\bot$ *and* $\top$ *stand for the empty and the universal relation on* $\mathbb{N}$*, respectively.*

Running example 2. For the sake of brevity, we only present the invariant difference sets and relations that are relevant to prove the pair candidate unreachable. So, $DS_{0,1}^{tk_0,tk_1} = \{0\}$, $DS_{1,2}^{tk_1,tk_2} = \{1\}$, $DS_{2,2}^{tk_2,tk_0} = \{1\}$, $\oplus_{0,1}^{tk_0,tk_1}$ is $=$, $\oplus_{1,2}^{tk_1,tk_2}$ is $>$, and $\oplus_{2,2}^{tk_2,tk_0}$ is $>$. □

We formalise this technique as follows. Simply put, we find values N_i, where N_i represents the value agreed by components as the number of times they applied rule i, such that they respect the relations we calculate for components.

Definition 13. *Let* $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ *be a supercombinator machine,* $(S, \Sigma, \Delta, \hat{s})$ *its induced LTS. For* $s = (s_{j(1)}, \ldots, s_{j(n)}) \in S$*:*

$$reachable_N(s) \mathrel{\widehat{=}} \exists\, N_1, \ldots, N_{|\Sigma|} \in \mathbb{N} \bullet \bigwedge_{i \in \{1 \ldots n\}} RC(i, j(i))$$

where:

- $RC(i, j(i)) \mathrel{\widehat{=}} \bigwedge_{\substack{k,l \in \Sigma \wedge \\ i \in pts(k) \cap pts(l)}} \begin{cases} True & \textit{if}\ \oplus_{i,j(i)}^{k,l} = \top \\ False & \textit{if}\ \oplus_{i,j(i)}^{k,l} = \bot \\ N_k \oplus_{i,j(i)}^{k,l} N_l & otherwise \end{cases}$

This predicate is false if either one the component states is trivially unreachable in its own component projection, for which $\oplus_{i,j(i)}^{k,l} = \bot$, or if all component states are trivially reachable but there exists an inconsistency on the RCs (Relation Constraints) calculated that shows that components cannot agree on the number of times they performed some rules. Either way, the snapshot must be unreachable.

Running example 2. Given the relations calculated, from $RC(0,1)$ we derive that $N_{tk_0} = N_{tk_1}$, from $RC(1,2)$ that $N_{tk_1} > N_{tk_2}$, and from $RC(2,2)$ that $N_{tk_2} > N_{tk_0}$. So, we can deduce that $N_{tk_0} = N_{tk_1}$ and $N_{tk_0} > N_{tk_1}$, a contradiction that shows that components cannot agree on the number of times they perform these rules and that $reachable_N((s_1, s_2, s_2))$ does not hold. Note this technique can show that the blocked state is unreachable for any such system with M ($M > 1$) components of which m ($M > m > 0$) hold initially a token. □

Given that components must synchronise on shared rules to reach snapshots, for any reachable snapshot, components must be able to, in particular, agree on the number of times they perform shared rules. So:

Theorem 2. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine with $(S, \Sigma, \Delta, \hat{s})$ its induced LTS. For $s \in S$, $reachable(s) \Rightarrow reachable_N(s)$.*

Thus, this predicate conservatively over-approximates reachability, and as such, it can be soundly used for deadlock-freedom analysis.

4.3 Abstraction

We can extend and improve these techniques by carrying out some abstractions. Firstly, observe that single-participant rules are irrelevant in our reachability analysis, as our techniques are based on the search of an inconsistency in the way components collaborate to reach a snapshot.

Secondly, we can achieve a sort of data abstraction for our techniques as follows. Intuitively, the application of a rule can be seen as a communication taking place between participants in this rule, whereas a set of rules involving the same exact participants might be seen as a set of possible values that they can communicate. With this view in mind, if we identify rules with the same participants, we are abstracting away these values and focusing on the fact a communication occurred between these participants. Our concrete framework and this abstract one can be seamlessly and uniformly integrated in our techniques by using the following partitioning and slightly modified component projection.

Definition 14. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine and $(S, \Sigma, \Delta, \hat{s})$ its induced LTS. For a given rule identifier $i \in \Sigma$, we have the following partitions:*

- *Concrete: $[i]_C \mathrel{\widehat{=}} i$*
- *Abstract: $[i]_A \mathrel{\widehat{=}} \boldsymbol{min}(\{j \mid j \in \Sigma \wedge \bullet\ pts(i) = pts(j)\})$ (where* **min** *returns the smallest integer in a non-empty finite set)*

We analyse slightly different component projections, depending on the level of abstraction we want.

Definition 15. *Let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine, where $L_i = (S_i, \Sigma_i, \Delta_i, \hat{s}_i)$, and $(S, \Sigma, \Delta, \hat{s})$ its induced LTS, and $x \in \{A, C\}$ a level of abstraction. The projection of $\mathcal{S}$ over component i is given by the supercombinator machine $\mathcal{S}_i = (\langle L_i \rangle, \{([j]_x, (e_i), a) \mid \exists (j, (e_1, \ldots, e_n), a) \in \mathcal{R} \bullet e_i \neq -\})$.*

So, we end up with two different predicates for each technique: $reachable^C_N(s)$ and $reachable^C_S(s)$ represent our original predicates, while $reachable^A_N(s)$ and $reachable^A_S(s)$ their abstract counterparts.

4.4 Discussion

Our frameworks are intended to automate some common methods for proving that a snapshot is unreachable. Some methods use the recent behaviour of components to show that they cannot cooperate to reach a system's snapshot [13, 12], while other methods rely on relational invariants to characterise states and prove snapshots unreachable [8, 17]. As both of our running examples show, we provide a fully systematic framework to carry out these specific sorts of reasoning.

Our reachability tests were inspired by Martins's CSDD and FSDD, which were in turn inspired by proof rules from [17]. We have, however, removed some of FSDD and CSDD's limitations. In particular, we propose reachability criteria that are completely independent of the safety property that is being checked, while both the CSDD and FSDD are centred on deadlock analysis.

5 Combining Reachability Tests with Local Analysis

In this section we combine the Pair characterisation, proposed in [2], with the new reachability tests presented in Section 4. In this new framework, a potential deadlock is a pair candidate that meets our new reachability tests.

Definition 16. *Let $\mathcal{S}$ be a supercombinator machine and $(S, \Sigma, \Delta, \hat{s})$ its induced LTS. A snapshot $s \in S$ is a* deadlock candidate *iff the following predicate holds:*

$$deadlock_candidate(s) \mathrel{\widehat{=}} pair_candidate(s) \wedge reachable^C_N(s) \wedge reachable^C_S(s) \\ \wedge reachable^A_N(s) \wedge reachable^A_S(s)$$

Given that our reachability tests over-approximate reachability and that every deadlock is also a pair candidate [2], every deadlock must also be a deadlock candidate. So, a system free of deadlock candidates has to be deadlock free.

Theorem 3. *If a supercombinator machine is deadlock-candidate free, then it must also be deadlock free.*

Our new characterisation is clearly more precise than the Pair one, but it remains imprecise: a blocked snapshot can be unreachable and yet meet all the imprecise reachability tests proposed. Nevertheless, by conjoining these new tests, we tighten the snapshot space analysed. Observe that it only takes one failed reachability test, out of the four proposed, to consider a snapshot unreachable. The incompleteness of our method is illustrated by the following example.

Example 1. Let $\mathcal{S} = (\langle L_1, L_2, L_3 \rangle, \mathcal{R})$ be the supercombinator machine such that L_1, L_2 and L_3 are described in Figure 3 and $\mathcal{R}$ requires components to synchronise on shared events. The snapshot (p_0, q_0, r_3) is blocked and it meets all reachability tests, but it is not reachable. Thus, it constitutes a deadlock candidate but not a deadlock. Neither local analysis nor the underlying proof methods in our reachability tests are strong enough to prove this snapshot unreachable. □

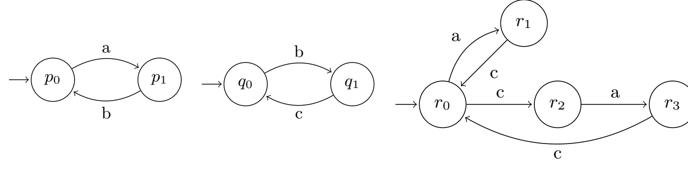

Fig. 3. LTSs of components L_1, L_2 and L_3, respectively.

5.1 Implementation

We built upon [2] to create an efficient implementation for our framework. So, we encode the search for a deadlock candidate as a satisfiability problem to be later checked by a SAT solver. For the remainder of this section, let $\mathcal{S} = (\langle L_1, \ldots, L_n \rangle, \mathcal{R})$ be a supercombinator machine, $(S, \Sigma, \Delta, \hat{s})$ its induced LTS, $\mathcal{S}_i$ the projection of $\mathcal{S}$ on component i, and $(S_i, \Sigma_i, \Delta_i, \hat{s}_i)$ its induced LTS.

In our propositional encoding, $s_{i,j}$ is the boolean variable representing the state s_j of component i, and $\mathcal{U}$ represents the disjoint union of all S_i sets. The assignment $s_{i,j} = true$ indicates this component state belongs to a deadlock candidate, whereas $s_{i,j} = false$ means it does not. Our formula $\mathcal{F} \mathrel{\widehat{=}} PC \wedge Reach_N^C \wedge Reach_S^C \wedge Reach_N^A \wedge Reach_S^A$ is a conjunction of five sub-formulas, each of them captures a predicate of our deadlock characterisation. The combination of component states assigned to true in a satisfying assignment of $\mathcal{F}$ forms a deadlock candidate.

The first sub-formula PC captures the pair-candidate characterisation; we reuse the propositional formula that is presented in [2]. The component states assigned to true in a satisfying assignment for PC form a pair-candidate snapshot.

Next, we present a way to encode our newly proposed reachability tests. First, we present how to encode the predicates $reachable_N^x$ for $x \in \{A, C\}$.

$$Reach_N^x \mathrel{\widehat{=}} \bigwedge_{s_{i,j} \in \mathcal{U}} s_{i,j} \Rightarrow RC(i,j)$$

We encode the variables $N_1, \ldots, N_{|\Sigma|}$ as bit-vectors of size $\lceil \log_2 |\Sigma| \rceil$, as we need $|\Sigma|$ distinct values to find a model for such a constraint[2]. We encode $<$, $=$ and $>$ as the corresponding operations on bit-vectors.

As follows, we present how to encode the predicates $reachable_S^x$ for $x \in \{A, C\}$. Let $occurs = \{P_b^a, \ldots, P_z^y\}$ with $occurs \mathrel{\widehat{=}} \bigcup\{\mathbf{SET}(SFO_{i,j}) \mid s_{i,j} \in \mathcal{U}\}$.

$$Reach_S^x \mathrel{\widehat{=}} \bigwedge_{s_{i,j} \in \mathcal{U}} s_{i,j} \Rightarrow HBC(i,j)$$

We encode the variables $clk_b^a, \ldots, clk_z^y$ as bit-vectors of size $\lceil \log_2 |occurs| \rceil$, again we only need $|occurs|$ distinct values to satisfy this formula[2]. We encode $<$ as the corresponding operation on bit-vectors.

[2] The cases where $|\Sigma| = 1$ or $|occurs| = 1$ are trivially possibly-reachable.

The rationale behind these two last sub-formulas is as follows. If the component state is assigned to true in a satisfying assignment, we make sure, by the implication, that the associated reachability constraint is also met. So, any satisfying-assignment snapshot has to meet our reachability tests.

6 Practical Evaluation

We here evaluate our new framework. FDR3's ability to analyse CSP and generate supercombinator machines is exploited in generating our SAT encoding, which is then checked by the Glucose 4.0 solver [7]. We call this new tool *DeadlOx*. A prototype of our DeadlOx and the models used in this section are available at [1]. For this experiment, we checked deadlock freedom for some CSP benchmark problems. The experiment was conducted on a dedicated machine with a quad-core Intel Core i5-4300U CPU @ 1.90GHz, and 8GB of RAM. We compare our prototype against: SDD, CSDD and FSDD (which are implemented in Martin's Deadlock Checker tool [14]); Pair technique [2]; FDR3's built-in deadlock freedom assertion [9], and its combination with partial order reduction (FDRp) [10] or compression techniques (FDRc) [18].

We analyse 13 systems that are deadlock free and triple disjoint. Out of these systems, 12 can be proved deadlock free by DeadlOx, 6 can be proved by CSDD, and 5 can be proved by FSDD. The latter two frameworks combine to prove 7 of the 13 systems deadlock free. Pair proves 6 of them deadlock-free, and SDD only 4 of them. The systems that we evaluated are: the alternating bit protocol (ABP), the butler solution to the dining philosophers (Butler), a distributed database (DDB), a matrix multiplication system (Matmul), the asymmetric solution to the dining philosophers (Phils), a ring network (Ring), the mad postman routing algorithm (Rout), the sliding window protocol (SWP), Milner's scheduler (Scheduler), a telephone switch system (Tel), a token ring system with a single token (Token Ring), a token ring system with $N/2$ tokens (Token Ring HF) and a train track system. These problems are discussed in detail in [19]. Table 1 presents the results that we obtain for 12 of the 13 systems; the train track system is not presented in this table as none of the incomplete techniques evaluated here can prove it deadlock free. DeadlOx fails on this example because neither of the additional reachability arguments are sufficient for this system; it seems to require invariants based explicitly on the number of tokens (i.e. trains), and the movement of the tokens is too unpredictable to capture using our rules.

For the benchmark problems analysed, DeadlOx is significantly more accurate than the other incomplete techniques (i.e. SDD, Pair, CSDD, and FSDD) while faring similarly in terms of analysis time. Comparing to the complete approaches (i.e. FDR3, FDR3c, FDR3p), DeadlOx is consistently faster than the best complete approach, which is the combination of FDR3's deadlock assertion with compression techniques, while being able to prove deadlock freedom for all the benchmark problems except for the train track example. We point out, however, that the effective use of compression techniques requires a careful and skilful application of those, whereas our method is fully automatic.

Example	N	Incomplete					Complete		
		DeadlOx	SDD	Pair	CSDD	FSDD	FDR3c	FDR3p	FDR3
ABP	50	**0.06**	0.27	0.06	0.28	0.29	+	0.13	0.17
	100	**0.07**	0.71	0.07	0.62	0.75	+	0.23	0.39
	200	**0.12**	1.89	0.12	1.95	1.97	+	0.60	1.29
Butler	5	**0.06**	-	0.06	-	-	0.10	0.07	0.07
	10	**0.36**	-	0.37	-	-	0.46	1.36	116.93
	12	**1.75**	-	1.72	-	-	1.30	12.78	*
	15	**19.57**	-	22.10	-	-	13.79	*	*
DDB	5	**0.15**	-	-	-	-	0.31	0.41	0.13
	10	**1.61**	-	-	-	-	*	*	*
	20	**56.39**	-	-	-	-	*	*	*
Matmul	5	**0.20**	-	-	0.11	-	0.16	0.07	*
	10	**3.66**	-	-	0.16	-	15.27	0.32	*
	20	**48.08**	-	-	0.59	-	*	22.18	*
	30	*	-	-	1.97	-	*	*	*
Phils	20	**0.07**	0.16	0.07	0.16	0.16	0.27	0.14	*
	50	**0.11**	0.23	0.13	0.23	0.23	1.42	0.75	*
	100	**0.18**	0.35	0.30	0.36	0.35	13.20	5.50	*
	500	**1.72**	2.78	5.42	2.80	2.80	*	*	*
Ring	50	**0.10**	-	-	-	0.13	0.29	*	*
	100	**0.15**	-	-	-	0.16	0.60	*	*
	200	**0.27**	-	-	-	0.28	1.41	*	*
	500	**0.81**	-	-	-	0.83	5.87	*	*
Rout	5	**0.10**	0.13	0.12	0.15	0.15	0.19	*	*
	10	**0.28**	0.30	0.99	0.32	0.31	0.68	*	*
	20	**2.05**	1.1	14.06	1.31	1.19	4.14	*	*
	50	**24.45**	21.5	*	23.05	22.30	115.36	*	*
SWP	3	**0.15**	0.91	0.14	0.93	0.90	0.24	0.21	2.9
	5	**3.52**	*	3.20	*	*	4.58	41.9	41.81
	7	**107.69**	*	105.69	*	*	136.64	*	*
Scheduler	100	**0.13**	-	-	0.15	-	0.29	0.43	*
	500	**0.57**	-	-	0.40	-	2.32	106.26	*
	1000	**1.36**	-	-	0.86	-	8.14	*	*
	1500	**2.43**	-	-	1.32	-	23.47	*	*
Tel	3	**0.06**	-	0.06	-	-	2.05	*	*
	5	**0.32**	-	0.32	-	-	*	*	*
	8	**2.88**	-	31.69	-	-	*	*	*
	10	**38.73**	-	*	-	-	*	*	*
Token Ring	15	**2.42**	-	-	-	-	+	5.62	0.34
	20	**11.95**	-	-	-	-	+	38.45	1.07
	25	**48.94**	-	-	-	-	+	171.52	2.97
Token Ring HF	15	**2.14**	-	-	-	-	+	*	*
	20	**11.63**	-	-	-	-	+	*	*
	25	**45.16**	-	-	-	-	+	*	*

Table 1. Benchmark efficiency comparison. N is a parameter that is used to alter the size of the system. We measure in seconds the time taken to check deadlock freedom for each system. * means that the method took longer than 300 seconds. - means that the method is unable to prove deadlock freedom. + means that no efficient compression technique could be found.

7 Conclusion

We combine the Pair imprecise characterisation given in [2] with two newly proposed reachability techniques to create a new framework for deadlock-freedom analysis. These new reachability techniques combine information extracted from static analysis of components with a global property of the system to show that components cannot cooperate to reach the snapshot under analysis. Our new framework is strictly more accurate than the Pair framework. Particularly, while Pair is unable to show that a snapshot is unreachable if that depends on a global aspect of the system, our new reachability tests can show that with respect to two specific global invariants of the system, namely, components have to agree on the order of cooperation and on the number of time they cooperate. Note, we only restrict this work to pairwise-communicating systems so we can re-use Pair's efficient strategy to encode the *blocked* predicate; our reachability tests and their encodings can be applied to systems with multiway communication. Moreover, the ideas in this paper should transfer easily to any formalism where systems are described by interacting LTSs.

We have implemented this new framework in the DeadlOx tool. This implementation shows that for the assessed benchmark systems, DeadlOx is substantially more accurate than similar incomplete techniques, whilst taking a similar amount of time to analyse systems. Also, as it seems to be consistently more efficient than complete techniques, it could be used as a preliminary step in deadlock-freedom checking. If it fails to prove deadlock freedom, then a complete method should be used. Note DeadlOx uses FDR3 to obtain supercombinator machines from systems described using CSP, but a tool analogous to DeadlOx could be created for other notations by replacing its use of FDR3 to generate such machines.

We plan to extend this work in two directions. Firstly, we would like to see how we could reuse (a part of) this framework to check other safety properties. In particular, we plan to reuse it to check trace-refinement properties and a notion of freedom from permanently blocked subsystems. Secondly, we plan to create additional imprecise tests for reachability, so we can have an even more accurate framework. Note, for instance, that our techniques are not strong enough to prove deadlock-freedom for one of the benchmark systems evaluated. We are particularly interested in the application of SAT solvers to infer system invariants.

Acknowledgements

The first author is a CAPES Foundation scholarship holder (Process no: 13201/13-1). The second and third authors are partially sponsored by DARPA under agreement number FA8750-12-2-0247. We thank the anonymous reviewers for their valuable comments.

References

1. Pedro Antonino, Thomas Gibson-Robinson, and A. W. Roscoe. Experiment package, 2016. `http://www.cs.ox.ac.uk/people/pedro.antonino/pkg.zip`.

2. Pedro Antonino, Thomas Gibson-Robinson, and A.W. Roscoe. Efficient deadlock-freedom checking using local analysis and SAT solving. In *IFM*, number 9681 in LNCS, pages 345–360. Springer, 2016.
3. Pedro Antonino, Marcel Medeiros Oliveira, Augusto Sampaio, Klaus Kristensen, and Jeremy Bryans. Leadership election: An industrial SoS application of compositional deadlock verification. In *NFM*, volume 8430 of *LNCS*, pages 31–45, 2014.
4. Pedro Antonino, A.W. Roscoe, and Thomas Gibson-Robinson. Tighter reachability criteria for deadlock-freedom analysis. Tech report, University of Oxford, 2016. `http://www.cs.ox.ac.uk/people/pedro.antonino/reach_techreport.pdf`.
5. Pedro Antonino, Augusto Sampaio, and Jim Woodcock. A refinement based strategy for local deadlock analysis of networks of CSP processes. In *FM*, volume 8442 of *LNCS*, pages 62–77, 2014.
6. Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. An Abstract Framework for Deadlock Prevention in BIP. In *FORTE*, number 7892 in LNCS, pages 161–177. Springer, 2013.
7. Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009.
8. Naiem Dathi. *Deadlock and Deadlock Freedom*. PhD thesis, University of Oxford, 1989.
9. Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A.W. Roscoe. FDR3 — A Modern Refinement Checker for CSP. In *TACAS*, volume 8413 of *LNCS*, pages 187–201, 2014.
10. Thomas Gibson-Robinson, Henri Hansen, A.W. Roscoe, and Xu Wang. Practical partial order reduction for CSP. In *NFM*, volume 9058 of *LNCS*, pages 188–203. Springer International Publishing, 2015.
11. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
12. Christian Lambertz and Mila Majster-Cederbaum. Analyzing Component-Based Systems on the Basis of Architectural Constraints. In *FSEN*, pages 64–79. Springer, April 2011.
13. Jeremy M. R. Martin. *The Design and Construction of Deadlock-Free Concurrent Systems*. PhD thesis, University of Buckingham, 1996.
14. J.M.R. Martin and S.A. Jassim. An efficient technique for deadlock analysis of large scale process networks. In *FME '97*, pages 418–441, 1997.
15. Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.
16. M. V. M. Oliveira, P. Antonino, R. Ramos, A. Sampaio, A. Mota, and A. W. Roscoe. Rigorous development of component-based systems using component metadata and patterns. *Formal Aspects of Computing*, pages 1–68, 2016.
17. A. W. Roscoe and Naiem Dathi. The pursuit of deadlock freedom. *Inf. Comput.*, **75(3)**:289–327, 1987.
18. A. W. Roscoe, Paul H. B. Gardiner, Michael Goldsmith, J. R. Hulance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In *TACAS*, pages 133–152, 1995.
19. A.W. Roscoe. *Understanding Concurrent Systems*. Springer, 2010.