Model Checking -Regular Properties with Decoupled Search

Gnad, Daniel; Eisenhut, Jan; Lluch Lafuente, Alberto; Hoffmann, Jörg

Published in:
Computer Aided Verification

Link to article, DOI:
10.1007/978-3-030-81688-9_19

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Model Checking ω-Regular Properties with Decoupled Search

Daniel Gnad[1⋆], Jan Eisenhut[1], Alberto Lluch Lafuente[2], and Jörg Hoffmann[1]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{gnad,hoffmann}@cs.uni-saarland.de;
s8jaeise@stud.uni-saarland.de
[2] Technical University of Denmark, Kongens Lyngby, Denmark
albl@dtu.dk

Abstract. Decoupled search is a state space search method originally introduced in AI Planning. Similar to partial-order reduction methods, decoupled search exploits the independence of components to tackle the state explosion problem. Similar to symbolic representations, it does not construct the explicit state space, but sets of states are represented in a compact manner, exploiting component independence. Given the success of both partial-order reduction and symbolic representations when model checking liveness properties, our goal is to add decoupled search to the toolset of liveness checking methods. Specifically, we show how decoupled search can be applied to liveness verification for composed Büchi automata by adapting, and showing correct, a standard algorithm for detecting lassos (i.e., infinite accepting runs), namely nested depth-first search. We evaluate our approach using a prototype implementation.

1 Introduction

Model checking is a well-known problem in formal verification. Given a formal description of a system $\mathcal{M}$, the model checking problem is to decide whether the system satisfies a property ϕ. In contrast to safety properties, which can only express whether there exists a finite run of the system that reaches a state with certain (bad) properties, liveness properties can express good behaviours of the system that should occur repeatedly, i.e., infinite runs in which something good happens infinitely often.

In this work, we consider a liveness verification problem that arises when composing a set $\mathcal{A}^1, \ldots, \mathcal{A}^n$ of *non-deterministic Büchi automata* (NBA), each with its own acceptance condition. We recall that an accepting run for a single NBA is a *lasso* $\rho_p(\rho_c)^\omega$ with a prefix ρ_p and a cycle ρ_c that visits an accepting state. For the composition of a set of NBAs into an NBA we consider the following liveness property: a composed run is accepting if there is a cycle visiting a state that is accepting for *all* components. Such a general problem captures standard liveness verification problems related to ω-regular properties. An archetypal example is automata-based LTL checking, where system components are represented as NBAs and are composed with a property monitor, represented as a Büchi automaton (often the negation of an LTL property). In this case an accepting composed run witnesses a violation of a linear-time property.

⋆ Corresponding author.

The predominant approach to address such verification problems using explicit state space search is to use *nested depth-first search* (NDFS) algorithms [5, 21, 31], also called *double depth-first search*, which perform on-the-fly checking of liveness properties while composing the NBAs. NDFS, like all state space search methods, suffers from the state explosion problem. Various methods, such as partial-order reduction [33, 26, 18, 10, 29], symbolic representations [2, 27], symmetry reduction [7, 22], or Petri-net unfolding [8, 9] have been proposed to alleviate the state explosion problem. Here, we add *decoupled state space search* [13], shortly *decoupled search*, as a new method for model checking liveness properties, complementary to the existing approaches. Indeed, as Gnad and Hoffmann [13, 14] have shown, decoupled search complements these techniques in the sense that there exist cases where it yields exponentially stronger reductions. It has also been shown that decoupled search can be fruitfully combined with partial-order reduction [15], symmetry reduction [17], and symbolic search [16].

Decoupled search has recently been introduced in AI planning [13], addressing goal reachability problems. Its applicability to model checking of safety properties has been shown in [12], where it was effectively introduced into the SPIN model checker [19]. However, the extension of decoupled search to cycle detection problems inherent to liveness model checking and NDFS algorithms has not yet been investigated. This paper addresses that investigation for the first time.

Decoupled search exploits the independence of system components, similar to partial-order reduction techniques, by not enumerating all interleavings of transitions across components. Similar to symbolic representations, decoupled search does not construct the explicit state space of the product. Instead, search nodes, called *decoupled states*, symbolically represent sets of states. Each decoupled state compactly represents many global states and their closure up to internal transitions of individual components. Similar to partial-order reduction or symbolic search, decoupled search can be exponentially more efficient than explicit search of the state space, as shown for reachability problems in the domains of AI planning [13] and model checking [12].

The main contribution of our paper is to extend the scope of decoupled search from safety properties, as done in [12], to liveness properties. In particular, we adapt a standard NDFS algorithm to the decoupled state representation. The resulting algorithms are able to solve the verification problem mentioned above, namely checking acceptance of composed NBAs. The main technical challenge for the correctness of our algorithms was to identify the conditions that imply existence of accepting runs in decoupled search and to show how such runs can be constructed efficiently.

We evaluate our decoupled NDFS algorithm using a prototype implementation on two showcase examples similar to the dining philosophers problem, and a set of randomly generated models. We compare to established tools, namely the SPIN model checker [19], and Petri-net unfolding with Cunf [29]. The results show that, like for safety properties, decoupled search can yield exponential advantages over state-of-the-art methods. In particular, its advantage grows with the degree to which components act independently of others, via internal transitions that do not affect other components.

The rest of the paper is structured as follows. We start in Section 2 by recalling the necessary background on NBAs, the verification problem we consider, and a standard NDFS algorithm typically used to solve the problem. Sections 3–5 present our contribu-

tion: Section 3 formalizes decoupled search in terms of composed NBAs, and shows its desired properties; Section 4 discusses some issues that would arise in a naïve attempt to (incorrectly) adapt it, and describes the (correct) adapted NDFS algorithm; Section 5 provides its correctness proof. In Section 6 we show our experimental evaluation, whose code and models are publicly available at [**?**]. Section 7 concludes the paper discussing related works and future research avenues.

2 Büchi Automata, Composition and Verification

This section recalls some basic notions of Büchi automata, their composition, the verification problem we consider in this paper for such composition, and its standard algorithmic resolution based on NDFS.

Büchi Automata and Accepting Runs. We start with the definition of non-deterministic Büchi automata (NBA).

Definition 1 (Non-Deterministic Büchi Automaton). *A* non-determinitic Büchi automaton $\mathcal{A}$ *is a tuple* $\langle S, \rightarrow, L, s_0, A\rangle$*, where* S *is a finite set of* states*,* L *is a finite set of* transition labels*,* $\rightarrow \subseteq S \times L \times S$ *is a* transition relation*,* $s_0 \in S$ *is an* initial state*, and* $A \in (S \rightarrow \mathbb{B})$ *is an* acceptance function*.*

A run ρ of an NBA is an infinite sequence of states $s_0, s_1, s_2, \cdots \in S^\omega$ starting from the initial state. The i-th state of a run ρ is denoted by $\rho[i]$ and we will use the same notation for other lists and sequences. A run ρ is accepting if it traverses accepting states infinitely often. Formally, $\overset{\infty}{\exists} j \in \mathbb{N} : A(s[j])$. We define a *trace* π of a run $\rho = s_0, s_1, s_2, \cdots \in S^\omega$ as a sequence of labels $\pi = l_0, l_1, \cdots \in L^\omega$ such that $\forall_{i\in\mathbb{N}} : \langle s_i, l_i, s_{i+1}\rangle \in \rightarrow$. We will also consider *finite* runs $\rho \in S^n$ and *finite* traces $\pi \in L^n$.

As hinted in Section 1, the existence of accepting runs is interesting for several theoretical and practical reasons. On the theoretical side, the language of an NBA is the set of all traces σ in L^ω for which an accepting run exists such that $\rho[i] \xrightarrow{\sigma[i]} \rho[i+1]$ for all $i \in \mathbb{N}$. On the practical side, model checking ω-regular properties, including LTL properties, can be reduced to checking the existence of accepting runs. Such runs, indeed, provide witnesses or counterexamples for the properties of interest.

Composition of NBAs. From now on we assume that the set of labels L of an NBA is partitioned into a set L_I of *internal labels* and a set L_G of *global labels*. The notion of composition we use is based on (maximal) synchronisation on global labels, in words: in every transition involving a global label, each component having the global label in its set of labels must perform a local transition, while transitions with internal labels can be performed independently. When composing NBAs we assume w.l.o.g. that they do not share any internal label. Further, we assume that every global label is shared by at least two component NBAs. Otherwise, such labels can be made internal. We will use the following notation: for a set $\mathcal{A}^1, \ldots, \mathcal{A}^n$ of NBAs, we use superscripting to denote the components of each $\mathcal{A}^i$, i.e., we assume $\mathcal{A}^i = \langle S^i, \rightarrow^i, L^i = L^i_I \cup L^i_G, s^i_0, A^i\rangle$.

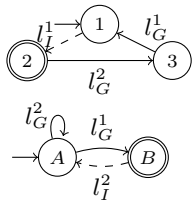
Definition 2 (Composition of NBAs). *The* composition *of n NBAs $\mathcal{A}^1, \ldots, \mathcal{A}^n$, denoted by $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$, is the NBA $\langle S, \to, L, \boldsymbol{s}_0, A\rangle$, where $S = S^1 \times \cdots \times S^n$, $L = \bigcup_{i\in\{1,\ldots,n\}} L^i$, $\boldsymbol{s}_0 = (s_0^1, \ldots, s_0^n)$, $A = \{(s_1, \ldots, s_n) \mapsto \wedge_{i=1,\ldots,n} A^i(s_i)\}$ and $\to$ is the smallest set of transitions closed under the following rules for interleaving of local transitions (1) and maximal synchronization on global labels (2):*

$$(1)\ \frac{s_i \xrightarrow{l_I} s_i' \qquad l_I \in L_I^i}{(s_1, \ldots, s_i, \ldots, s_n) \xrightarrow{l_I} (s_1, \ldots, s_i', \ldots, s_n)}$$

$$(2)\ \frac{\exists_{i\in\{1,\ldots,n\}} : l_G \in L_G^i \quad \forall_{j\in\{1,\ldots,n|l_G\in L_G^j\}} : s_j \xrightarrow{l_G} s_j' \quad \forall_{j\in\{1,\ldots,n|l_G\notin L_G^j\}} : s_j' = s_j}{(s_1, \ldots, s_n) \xrightarrow{l_G} (s_1', \ldots, s_n')}$$

As notation convention, we will denote component states simply by small case letters, e.g. s, and composed states $(s_1, \ldots, s_n) \in S$ by $\boldsymbol{s}$, i.e., as a vector, and similarly for local runs ρ (resp. traces π) and composed runs $\boldsymbol{\rho}$ (composed traces $\boldsymbol{\pi}$).

In Figure 1 we illustrate a small example of a composition of two NBAs $\mathcal{A}^1, \mathcal{A}^2$. In the top of the figure, we show the local state space of the two components ($\mathcal{A}^1$ left, $\mathcal{A}^2$ right), where the component states are $S^1 = \{1, 2, 3\}$, $S^2 = \{A, B\}$, and the labels are defined as $L_G^1 = L_G^2 = \{l_G^1, l_G^2\}$, $L_I^1 = \{l_I^1\}$, $L_I^2 = \{l_I^2\}$. A local state is accepting for $\mathcal{A}^1$, so $A^1(s) = \top$, iff $s = 2$, and similar $A^2(s) = \top$ iff $s = B$. The initial states are $s_0^1 = 1$ and $s_0^2 = A$. The transitions are as shown. In the bottom, we depict the part of the state space of the composition $\mathcal{A}^1 \parallel \mathcal{A}^2$ reachable from $\boldsymbol{s}_0 = (1, A)$ as it would be generated by a standard DFS. Here, transitions via global labels synchronize the components, internal transitions are executed independently. The states crossed out would be pruned by duplicate checking, the underlined state is accepting.



$$(1, A) \xrightarrow{l_I^1} (2, A) \xrightarrow{l_G^2} (3, A) \xrightarrow{l_G^1} (1, B) \begin{array}{l} \xrightarrow{l_I^1} \underline{(2, B)} \xrightarrow{l_I^2} \cancel{(2, A)} \\ \xrightarrow{l_I^2} \cancel{(1, A)} \end{array}$$

Fig. 1. Example of two NBAs, $\mathcal{A}^1$ and $\mathcal{A}^2$, and the state space of their composition $\mathcal{A}^1 \parallel \mathcal{A}^2$.

Verification problem and its resolution with NDFS. The verification problem we address in this paper is the existence of accepting runs in the composed NBA $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$. In words, we look for runs in $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ that infinitely often traverse states in which *all* component NBAs are in an accepting state. We discuss alternative acceptance conditions in Section 7.

Determining the existence of accepting runs in an NBA can be boiled down to the existence of so-called *lassos*, i.e., finite sequences of states in the NBA of the form $\boldsymbol{\rho}_p \boldsymbol{\rho}_c$

```
CheckEmptiness(A^1 || ... || A^n):
    Stack ← ⟨s_0⟩
    V ← ∅
    V' ← ∅
    DFS(s_0)
    return empty

NestedDFS(s):
    for all t s.t. s → t do
        if t ∈ V' then continue
        if t ∈ Stack then return cycle
        V' = V' ∪ {t}
        NestedDFS(t)

DFS(s):
    V = V ∪ {s}
    for all t s.t. s → t do
        if t ∈ V then continue
        push(Stack, t)
        DFS(t)
        pop(Stack)
    if A(s) then
        NestedDFS(s)
        V' = V' ∪ {s}
```

Fig. 2. A standard NDFS algorithm for lasso search in composed NBAs.

where $\boldsymbol{\rho}_p$ is the prefix of the lasso and $\boldsymbol{\rho}_c$ is the cycle of the lasso, which contains at least one accepting state and closes the cycle (i.e., $\boldsymbol{\rho}_p[|\boldsymbol{\rho}_p| - 1] = \boldsymbol{\rho}_c[|\boldsymbol{\rho}_c| - 1]$. Such a finite sequence of states represents an accepting run $\boldsymbol{\rho}_p(\boldsymbol{\rho}_c)^\omega$.

Several algorithms can be used to check the existence of lassos. The predominant family of algorithms are the variants of NDFS, originally introduced in [5]. Figure 2 shows the pseudo-code for one such variant, based on NDFS as presented in [4]. The algorithm is based on an ordinary depth-first search algorithm (**DFS**) that works as usual: a set V is used to record already visited states, and recursion enforces the depth-first exploration order of the state space. Moreover, a stack $Stack$ is used to keep track of the states on the current initial trace being explored. The main difference w.r.t. ordinary DFS is that a second, nested, depth-first search algorithm (**NestedDFS**) is invoked from accepting states on backtracking, i.e., after the recursive call to **DFS**. The idea is that, if this second depth-first search finds a state that is on $Stack$, then it is guaranteed that a cycle has been found, which contains at least one accepting state. That is, one finds the (un)desired lasso. The algorithm is also complete: no accepting cycle is missed.

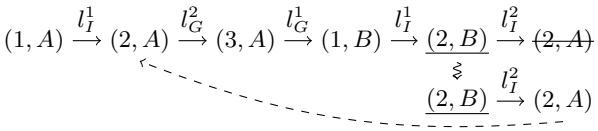


Fig. 3. Example run of **CheckEmptiness**. The wavy arrow indicates the invocation of **NestedDFS**$((2, B))$; the dashed arrow indicates how the cycle is closed.

In Figure 3, we illustrate an example run of the **CheckEmptiness** algorithm on our example. When **DFS** backtracks from $(2, B)$, **NestedDFS** is invoked, illustrated by the wavy arrow. **NestedDFS** generates the successor $(2, A)$, which is on *Stack*, so a cycle is reported. We can construct an accepting run $\boldsymbol{\rho}_p(\boldsymbol{\rho}_c)^\omega$ with prefix $\boldsymbol{\rho}_p$ induced by the trace l_I^1 and cycle $\boldsymbol{\rho}_c$ induced by the trace $l_G^2, l_G^1, l_I^1, l_I^2$.

3 The Decoupled State Space for Composed NBAs

As previously stated, decoupled state space search was recently developed in AI planning [13], and adapted to model checking of safety properties later on [12]. It is designed to tackle the state explosion problem inherent in search problems that result from compactly represented systems with exponentially large state spaces. In AI planning, where decoupled search was originally introduced, such systems are modelled through state variables and a set of transition rules (called "actions"). The adaptation of decoupled search to reachability checking in SPIN presented in [12] devised decoupled search for automata models, but informally only. Here, we introduce decoupled search formally for NBA models. We define the decoupled state space for composed NBAs, as the result from the composition of a set of NBAs.

3.1 Decoupled Composition of NBAs

In contrast to the explicit construction of the state space, where all reachable states are generated by searching over all traces of enabled transitions, decoupled search only searches over traces of global transitions, the ones that synchronize the component NBAs. In decoupled search, a *decoupled state* $s^{\mathcal{D}}$ compactly represents a set of states closed by internal steps. This is done in terms of the sequence of global labels used to reach these states, plus a set of reached states for each component. Definition 3 formalizes this through the operation *decoupled composition of NBAs*, which adapts the composition operation provided in Definition 2 to decoupled state space search.

Definition 3 (Decoupled composition of NBAs). *The* decoupled composition *of n NBAs $\mathcal{A}^1, \ldots, \mathcal{A}^n$, denoted by $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$, is a tuple $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L_G, s_0^{\mathcal{D}}, A^{\mathcal{D}} \rangle$ defined as follows:*

- $S^{\mathcal{D}} = \mathcal{P}^+(S^1) \times \cdots \times \mathcal{P}^+(S^n)$, *with* $\mathcal{P}^+(S) := 2^S \setminus \emptyset$.
- $s_0^{\mathcal{D}} = \langle \mathrm{iclose}(s_0^1), \ldots, \mathrm{iclose}(s_0^n) \rangle$, *with* $\mathrm{iclose}(s)$ *being the set of states s' that are reachable from s in $\mathcal{A}^i$ using only $\mathcal{A}^i$'s internal transitions L_I^i:*
  $\mathrm{iclose}(s) = \{s' \mid s \xrightarrow{l_I \in L_I^i}{}^* s'\}$ *and* $\mathrm{iclose}(S) = \bigcup_{s \in S} \mathrm{iclose}(s)$.
- $A^{\mathcal{D}}(s^{\mathcal{D}}) = \forall_{i \in \{1,\ldots,n\}} : \exists s^i \in S_i : A^i(s^i)$, *where* $s^{\mathcal{D}} = \langle S_1, \ldots, S_n \rangle$.
- $\rightarrow_{\mathcal{D}}$ *is the smallest set of transitions closed under the following rule:*

$$\frac{l_G \in L_G \quad \forall_{i \in \{1,\ldots n\}} : S_i' = \{s_i' \mid \boldsymbol{s} \in s^{\mathcal{D}} : \boldsymbol{s} \xrightarrow{l_G} (s_1', \ldots, s_i', \ldots, s_n')\} \quad S_i' \neq \emptyset}{s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} \langle \mathrm{iclose}(S_1'), \ldots, \mathrm{iclose}(S_n') \rangle}$$

where, abusing notation, we write $\boldsymbol{s} \in s^{\mathcal{D}}$ if $s^{\mathcal{D}} = \langle S_1, \ldots, S_n \rangle$ and $\boldsymbol{s} \in S_1 \times \ldots \times S_n$.

In the decoupled composition $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$ a decoupled state $s^{\mathcal{D}}$ is defined by a tuple $\langle s^{\mathcal{D}}[\mathcal{A}^1], \ldots, s^{\mathcal{D}}[\mathcal{A}^n] \rangle$, consisting of a non-empty set of component states $s^{\mathcal{D}}[\mathcal{A}^i]$ for each $\mathcal{A}^i$. A decoupled state represents exponentially many *member states*, namely all composed states $\boldsymbol{s} = (s_1, \ldots, s_n)$ such that $\boldsymbol{s} \in s^{\mathcal{D}}[\mathcal{A}^1] \times \cdots \times s^{\mathcal{D}}[\mathcal{A}^n]$. We will always use a superscript $\mathcal{D}$ to denote decoupled states $s^{\mathcal{D}}$.

We overload the subset operation $\subseteq$ for decoupled states $s^{\mathcal{D}}$ by doing it componentwise on the sets of reached local states, namely $s^{\mathcal{D}} \subseteq t^{\mathcal{D}} \Leftrightarrow \forall \mathcal{A}^i : s^{\mathcal{D}}[\mathcal{A}^i] \subseteq t^{\mathcal{D}}[\mathcal{A}^i]$.

During a search in the decoupled composition we define the *global trace* of a decoupled state $s^{\mathcal{D}}$, denoted $\pi^G(s^{\mathcal{D}})$, as the sequence of global transitions on which $s^{\mathcal{D}}$ was reached from $s_0^{\mathcal{D}}$. For DFS, as considered in this work, this is well-defined.

In explicit state search, states that have been visited before – duplicates – are pruned to avoid repeating the search effort unnecessarily. The corresponding operation in decoupled search is *dominance pruning* [13]. A newly generated decoupled state $t^{\mathcal{D}}$ is pruned if there exists a previously seen decoupled state $s^{\mathcal{D}}$ that *dominates* $t^{\mathcal{D}}$, i.e., where $t^{\mathcal{D}} \subseteq s^{\mathcal{D}}$. With the correctness result given below, this is safe. One can make the representation of decoupled states, and thereby also the dominance checking, more efficient by representing the state sets $s^{\mathcal{D}}[\mathcal{A}^i]$ symbolically [16].

The initial decoupled state is obtained by closing each local state with internal steps (iclose), and decoupled transitions generate decoupled states whose local states are also closed under internal steps. This maximally preserves the decomposition afforded by the decoupled representation. Namely, as we will prove in what follows, a decoupled state $s^{\mathcal{D}}$ compactly represents all explicit states that are reachable via traces that extend the global trace $\pi^G(s^{\mathcal{D}}) = l_G^1, l_G^2, \ldots, l_G^k$ with local transition labels. That is, for every component $\mathcal{A}^i$, $s^{\mathcal{D}}$ contains the non-empty subset of its local states $s^{\mathcal{D}}[\mathcal{A}^i] \subseteq S^i$ that can be reached with traces $\pi_i = l_1, l_2, \ldots, l_n$ such that there exist indices $j_1 < j_2 < \cdots < j_k$ where $l_{j_t} = \pi^G(s^{\mathcal{D}})[j_t]$ for all $1 \leq t \leq k$. In words, after every global label on $\pi^G(s^{\mathcal{D}})$, arbitrary enabled sequences of internal transitions are allowed.

We remark that the decoupled composition of a set of NBAs is always deterministic. For every pair of decoupled state $s^{\mathcal{D}}$ and global label l_G, there is a unique successor $t^{\mathcal{D}}$. This is easy to see, since if there is a composed state $\boldsymbol{s}$ contained in $s^{\mathcal{D}}$ that has multiple outgoing transitions labelled with l_G, all of the composed successor states are contained in $t^{\mathcal{D}}$. This increases the possible state space reduction compared to standard search, which needs to branch over all these successors. Note that this is different from the determinization of NBA, which comes with a blow-up [30]. The determinism is a consequence of the compact representation where all possible outcome states of a non-deterministic transition are contained in the decoupled successor state.

3.2 Correctness of Decoupled Composition

In this section we show that decoupled search, as presented here, is sound and complete w.r.t. reachability properties. We adapt the corresponding result from AI planning [13].

We require some additional notation. For a trace $\boldsymbol{\pi}$, by $\pi^G(\boldsymbol{\pi})$ we denote the subsequence of $\boldsymbol{\pi}$ that is obtained by projecting onto the global labels L_G.

As previously stated, the decoupled state space captures reachability of the composed system exactly. The proof is an adaptation of previous results from AI Planning [13] to composed NBAs as considered here.

Theorem 1. *A state $\boldsymbol{t}$ of a composition of NBAs $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ is reachable from a state $\boldsymbol{s}$ via a trace $\boldsymbol{\pi}$, iff there exist decoupled states $s^{\mathcal{D}}$, $t^{\mathcal{D}}$ in the decoupled composition $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$, such that $\boldsymbol{s} \in s^{\mathcal{D}}$, $\boldsymbol{t} \in t^{\mathcal{D}}$, and $t^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $\pi^G(\boldsymbol{\pi})$.*

Fig. 4. Illustration of the exponential separations to ample sets (left) and unfolding (right).

Proof. Let $\pi^G(\boldsymbol{\pi}) = l_G^1, \ldots, l_G^k$, and $s_i^{\mathcal{D}} \xrightarrow{l_G^{i+1}}_{\mathcal{D}} s_{i+1}^{\mathcal{D}}$ for all $1 \leq i < k$. We prove the claim by induction over the length of $\pi^G(\boldsymbol{\pi})$. For the base case $|\pi^G(\boldsymbol{\pi})| = 0$, the claim trivially holds, since, by the definition of $\mathrm{iclose}()$, $s^{\mathcal{D}}$ contains all composed states $\boldsymbol{t}$ that are reachable from any $\boldsymbol{s} \in s^{\mathcal{D}}$ via only internal transitions.

Assume a decoupled state $s_i^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $l_G^1, \ldots, l_G^i$. Then, by the definition of decoupled transitions and $\mathrm{iclose}()$, the state $s_{i+1}^{\mathcal{D}}$ contains all composed states $\boldsymbol{s}_{i+1}$ that are reachable from a state $\boldsymbol{s}_i \in s_i^{\mathcal{D}}$ via a trace $\pi^{i \to i+1}$ that consists of only internal transitions and l_G^{i+1}. By hypothesis, we can extend the traces reaching every such $\boldsymbol{s}_i$ from a $\boldsymbol{s} \in s^{\mathcal{D}}$ by $\pi^{i \to i+1}$ and obtain a trace reaching $\boldsymbol{s}_{i+1}$ from $\boldsymbol{s}$ with global sub-trace $l_G^1, \ldots, l_G^i, l_G^{i+1}$.

For the other direction, if a composed state $\boldsymbol{s}_i$ is reached in a decoupled state $s_i^{\mathcal{D}}$ and can reach a state $\boldsymbol{s}_{i+1}$ via a trace $\pi^{i \to i+1}$ that consists of internal labels and l_G^{i+1}, then there exists a decoupled transition $s_i^{\mathcal{D}} \xrightarrow{l_G^{i+1}}_{\mathcal{D}} s_{i+1}^{\mathcal{D}}$ and, again by the definition of decoupled transitions and $\mathrm{iclose}()$, $s_{i+1}^{\mathcal{D}}$ contains $\boldsymbol{s}_{i+1}$. By hypothesis $s_i^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$, where $\boldsymbol{s}_i$ is reachable from $\boldsymbol{s} \in s^{\mathcal{D}}$. Thus, $s_{i+1}^{\mathcal{D}}$ is reachable from $s^{\mathcal{D}}$ via $l_G^1, \ldots, l_G^i, l_G^{i+1}$. □

3.3 Relation to other State-Space Reduction Methods

Prior work has investigated the relation of decoupled search to other state-space reduction methods in the context of AI planning [13, 14], in particular to strong stubborn sets [33], Petri-net unfolding [8, 9], and symbolic representations using BDDs [2, 27]. For all these techniques, there exist families of scaling examples where decoupled search is exponentially more efficient.

We capture this formally in terms of *exponential separations*. A search method X is *exponentially separated* from decoupled search if there exists a family of models $\{M^n{=}\mathcal{A}^1, \ldots, \mathcal{A}^m \mid n \in \mathbb{N}\}$ of size polynomially related to n such that (1) the number of reachable decoupled states in $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^m$ is bounded by a polynomial in n, and (2) the state space representation of $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^m$ under X is exponential in n.

We next describe two scaling models showing that the ample sets variant of SPIN [20, 28], as a representative for partial-order reduction in explicit-state search, and Petri-net unfolding are exponentially separated from decoupled search. For symbolic search with BDDs, the reduction achieved by both methods is in general incomparable.

For ample sets, a simple model family looks as follows: there are n components, each with the same state space: two local states A_i, B_i, initial state A_i, two global transitions $l_G^{a,i}$, $l_G^{b,j}$, one internal transition. A component and the transitions are depicted in

the left of Figure 4 (the dashed transition is internal). The global transitions synchronize components pairwise; our argument holds for every possible such synchronization.

Under ample set pruning, no reduction is achieved (no state is pruned) because there is a global transition enabled in every state. Thus, there exists no state where only safe (i.e. internal) transitions are enabled, and the search always branches over all enabled transitions of all components. The decoupled state space, in contrast, only has a single decoupled state, where both local states are reached in each component. All decoupled successor states are dominated and will be pruned.

Similar to decoupled search, Petri-net unfolding exploits component independence by a special representation. Instead of searching over composed states and pruning transitions, the states of individual components are maintained separately.[3]

A scaling model showing that unfolding is exponentially separated from decoupled search is illustrated in the right of Figure 4. There are n components, each with the same state space with three local states A_i, B_i, C_i, a global label l_G, and transitions as shown in the figure. In a Petri net, this model is encoded with $3n$ places and 2^n transitions, one for every combination of one output place in each of the components. In the unfolding, this results in an event (the equivalent of a state) for every net transition. The decoupled state space has only two decoupled states: the initial state where $\{A_i\}$ is reached for all components, and its l_G-successor where $\{B_i, C_i\}$ is reached in every component.

4 NDFS for Decoupled Search

We now adapt NDFS to decoupled search. We start by discussing the deficiencies of a naïve adaptation. We will then introduce the key concepts in our fixed algorithm in Section 4.2, and present the algorithm itself in Section 4.3. We close this section by showing that the exponential separations to partial-order reduction and unfolding from Section 3.3 carry over to liveness checking by simple modifications of the models.

4.1 Issues with a Naïve Adaptation of NDFS

In a naïve adaptation of NDFS to decoupled search, the only thing that changes is the treatment of decoupled states, which represent *sets of composed states*, compared to single states in the standard variant. This leads to three mostly minor changes: (1) instead of duplicate checking we perform dominance pruning; (2) checking if a decoupled state is accepting boils down to checking if it contains an accepting member state; and (3) to see if a state $\boldsymbol{t}$ contained in a state $t^{\mathcal{D}}$ generated in **NestedDFS** is on the stack, we need to check if $t^{\mathcal{D}}$ has a non-empty intersection with a state on $Stack$.

As we will show next, it turns out that this naïve adaptation can *miss* cycles due to pruning. Revisiting a composed state in **NestedDFS** does actually not imply a cycle, because reaching $t^{\mathcal{D}}$ from $s^{\mathcal{D}}$ entails only that every member state of $t^{\mathcal{D}}$ can be reached from *at least one* member state of $s^{\mathcal{D}}$, not from all of them. The critical point is that pruning does not take into account *from where* states are reachable.

[3] A general difference between the methods is that checking reachability of a conjunctive property is linear in the number of decoupled states, but **NP**-complete for an unfolding prefix [26].

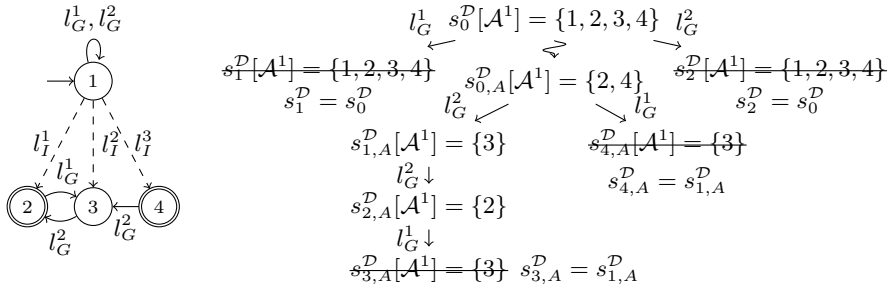


Fig. 5. Counterexample showing that a naïve adaptation of the NDFS algorithm is incomplete. The (only) component NBA $\mathcal{A}^1$ is depicted on the left. The search tree on the right shows the entire reachable decoupled state space, where pruned states are crossed out; the wavy arrow depicts the invocation of **NestedDFS** on the acceptance restriction $s_{0,A}^{\mathcal{D}}$ of $s_0^{\mathcal{D}}$.

Consider the example in Figure 5. The left part of the figure shows the local state space of component NBA $\mathcal{A}^1$. For simplicity, we only show a single component, which is sufficient to illustrate the issue. Here, $\mathcal{A}^1$ is defined as follows: $S^1 = \{1,2,3,4\}$, $L_G^1 = \{l_G^1, l_G^2\}$, $L_I^1 = \{l_I^1, l_I^2, l_I^3\}$, $A^1(s) = \top$ iff $s \in \{2,4\}$, and $s_0^1 = 1$. The transitions are as shown in the left of the figure. The decoupled search space generated using NDFS is depicted in the right of the figure. Pruned states are crossed out.

NestedDFS is launched (indicated by the wavy arrow) on the accepting initial state $s_0^{\mathcal{D}}$. Before explaining the main issue, we remark that, to ensure that a cycle through an *accepting* member state of $s_0^{\mathcal{D}}$ is found, not a cycle through a non-accepting one, we need to restrict the set of reached local states to those that are accepting, and the states internally reachable from those via $\mathrm{iclose}()$. Thus, **NestedDFS** starts in what we call the acceptance-restriction $s_{0,A}^{\mathcal{D}}$ of $s_0^{\mathcal{D}}$, where $s_{0,A}^{\mathcal{D}}[\mathcal{A}^1] = \{2,4\}$. Now, the issue results from the fact that $s_{0,A}^{\mathcal{D}}$ contains two accepting member states, only one of which, namely state 2, is on a cycle. Assuming that the decoupled states are generated in order of increasing subscripts, so $s_1^{\mathcal{D}}$ before $s_2^{\mathcal{D}}$ and so on, state 2 is first reached in **NestedDFS** as a member state of $s_{2,A}^{\mathcal{D}}$, but via the transition labelled with l_G^2 from state 3, so the cycle cannot be closed. When generating the l_G^1 successor $s_{4,A}^{\mathcal{D}}$ of $s_{0,A}^{\mathcal{D}}$, its only member state 3 has already been reached in $s_{1,A}^{\mathcal{D}}$, so $s_{4,A}^{\mathcal{D}}$ is pruned and the cycle of state 2 via l_G^1, l_G^2 is missed. In the next Section we show how to fix this, through an extended state representation that keeps track of reachability from a set of reference states.

Another minor issue are lassos $\boldsymbol{\rho}_p(\boldsymbol{\rho}_c)^\omega$ whose cycle $\boldsymbol{\rho}_c$ is induced by internal labels only. These will not be detected, because **NestedDFS** only considers traces via global labels. We fix this by checking for L_I-cycles in every accepting decoupled state generated during **DFS**, to see if there exists a component that can reach such a state.

4.2 Reference-State Splits

The problem underlying the issue described in the previous section is that pruning is done regardless of the accepting states in the root node of **NestedDFS**. We now introduce an operation on decoupled states splitting them with respect to the set of reached

local accepting states for each component. In our algorithm, this will serve to distinguish the different accepting states, and thus force dominance pruning to distinguish reachability from these. Formally, we define the restriction to accepting local states as a new transition with a global label l_G^A that is a self-loop for all accepting states:

Definition 4 (Acceptance-Split Transition). *Let $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L, s_0^{\mathcal{D}}, A^{\mathcal{D}} \rangle$ be the decoupled composition of $\mathcal{A}^1, \ldots, \mathcal{A}^n$. Let $s^{\mathcal{D}}$ be an accepting decoupled state, and for $1 \leq i \leq n$ let $\langle s_1^i, \ldots, s_{c_i}^i \rangle \subseteq s^{\mathcal{D}}[\mathcal{A}^i]$ be the list of reached accepting states of $\mathcal{A}^i$, where for all $1 \leq j \leq c_i : A^i(s_j^i) = \top$. Then the* acceptance-split transition l_G^A *in $s^{\mathcal{D}}$ is defined as follows:*

$$\frac{A^{\mathcal{D}}(s^{\mathcal{D}}) = \top \quad \forall i \in \{1, \ldots n\}, j \in \{1, \ldots, c_i\} : s_j^i \in s^{\mathcal{D}}[\mathcal{A}^i] \wedge A^i(s_j^i) = \top}{s^{\mathcal{D}} \xrightarrow{l_G^A}_{\mathcal{D}} \langle\langle \mathrm{iclose}(s_1^1), \ldots, \mathrm{iclose}(s_{c_1}^1)\rangle, \ldots, \langle \mathrm{iclose}(s_1^n), \ldots, \mathrm{iclose}(s_{c_n}^n)\rangle\rangle}$$

The outcome state $s_A^{\mathcal{D}}$ of an acceptance-split transition is a split decoupled state*. The set of* reference states *of $s_A^{\mathcal{D}}$ is $R(s_A^{\mathcal{D}}) := \{s \mid \exists \mathcal{A}_i : s \in s^{\mathcal{D}}[\mathcal{A}^i] \wedge A^i(s) = \top\}$.*

In words, the operation splits up the single set of reached component states $s^{\mathcal{D}}[\mathcal{A}^i]$ of $\mathcal{A}^i$ into a list of state sets, where each such set $s_A^{\mathcal{D}}[\mathcal{A}^i]_s$ contains the states that can be reached via internal transitions from the respective accepting state $s \in s^{\mathcal{D}}[\mathcal{A}^i]$.

Our search algorithm will use the acceptance-split transition to generate the root node $s_A^{\mathcal{D}}$ of **NestedDFS** from an accepting state $s^{\mathcal{D}}$ backtracked from in **DFS**. Hence **NestedDFS** will search in the space of split decoupled states. The transitions over these behind an $s_A^{\mathcal{D}}$ are defined as follows:

Definition 5 (Split Transitions). *Let $\langle S^{\mathcal{D}}, \rightarrow_{\mathcal{D}}, L, s_0^{\mathcal{D}}, A^{\mathcal{D}} \rangle$ be the decoupled composition of $\mathcal{A}^1, \ldots, \mathcal{A}^n$. Let $s^{\mathcal{D}}$ and $t^{\mathcal{D}}$ be decoupled states, with a transition $s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}$. Let $\langle s_1^i, \ldots, s_{c_i}^i \rangle \subseteq S^i$ be reference states for $\mathcal{A}^i$. Then the* split transition $s_R^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t_R^{\mathcal{D}}$ *is defined such that for every $\mathcal{A}^i$ and every $1 \leq j \leq c_i$ we have:*

$$t_R^{\mathcal{D}}[\mathcal{A}^i]_{s_j^i} = \begin{cases} \mathrm{iclose}(\{s' \in t^{\mathcal{D}}[\mathcal{A}^i] \mid \exists s \in s_R^{\mathcal{D}}[\mathcal{A}^i]_{s_j^i} : s \xrightarrow{l_G}{}^i s'\}) & l_G \in L_G^i \\ s_R^{\mathcal{D}}[\mathcal{A}^i]_{s_j^i} & l_G \notin L_G^i \end{cases}$$

The list of reference states for an $\mathcal{A}_i$ does not change along a trace of split transitions. Let $s_A^{\mathcal{D}}$ be a decoupled state generated by an acceptance-split transition $s^{\mathcal{D}} \xrightarrow{l_G^A}_{\mathcal{D}} s_A^{\mathcal{D}}$, then for all successor states $t^{\mathcal{D}}$ of $s_A^{\mathcal{D}}$, the set of reference states is $R(t^{\mathcal{D}}) = R(s_A^{\mathcal{D}})$.

We extend set operations to the split representation as follows. A split decoupled state $s_R^{\mathcal{D}}$ *dominates* a split decoupled state $t_R^{\mathcal{D}}$, denoted $t_R^{\mathcal{D}} \subseteq_R s_R^{\mathcal{D}}$, if $R(t_R^{\mathcal{D}}) \subseteq R(s_R^{\mathcal{D}})$ and for all components $\mathcal{A}^i$ and reference states $s \in R(t_R^{\mathcal{D}}) \cap S^i$ we have $t_R^{\mathcal{D}}[\mathcal{A}^i]_s \subseteq s_R^{\mathcal{D}}[\mathcal{A}^i]_s$. In contrast, state membership is defined in a global manner, across reference states. Namely, the set of local states of an $\mathcal{A}^i$ reached in a split decoupled state $s_R^{\mathcal{D}}$ is defined as $s_R^{\mathcal{D}}[\mathcal{A}^i] := \bigcup_{s \in R(s_R^{\mathcal{D}}) \cap S^i} s_R^{\mathcal{D}}[\mathcal{A}^i]_s$. Composed state membership is defined relative to these $s_R^{\mathcal{D}}[\mathcal{A}^i]$ as before.

An important property of the splitting is that it preserves reachability of member states. Concretely, for a split-transition $s_R^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t_R^{\mathcal{D}}$ induced by a transition $s^{\mathcal{D}} \xrightarrow{l_G}_{\mathcal{D}} t^{\mathcal{D}}$ for all $\mathcal{A}^i$ it holds that if $s_R^{\mathcal{D}}[\mathcal{A}^i] = s^{\mathcal{D}}[\mathcal{A}^i]$, then $t_R^{\mathcal{D}}[\mathcal{A}^i] = t^{\mathcal{D}}[\mathcal{A}^i]$.

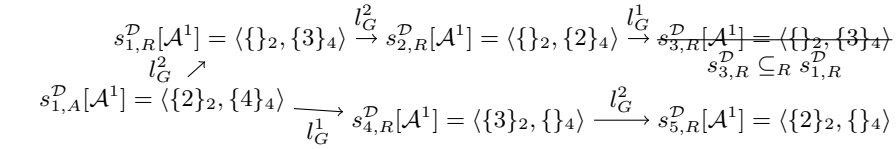


Fig. 6. With acceptance-splitting, **NestedDFS** invoked on the l^A_G-successor $s^{\mathcal{D}}_{1,A}$ of $s^{\mathcal{D}}_0$ of the example in Figure 5 correctly detects the cycle of state 2 induced by the trace l^1_G, l^2_G.

As a notation convention, we will always denote split states $s^{\mathcal{D}}_R$ by a subscript R, and the direct outcome of an acceptance-split transition by $s^{\mathcal{D}}_A$, with a subscript A.

Considering our example again, Figure 6 illustrates how, on split decoupled states, the cycle $2 \xrightarrow{l^1_G} 3 \xrightarrow{l^2_G} 2$ is not pruned. The state $s^{\mathcal{D}}_{3,R}$ is still pruned, as it contains only component states reached from state 4. In $s^{\mathcal{D}}_{4,R}$ and $s^{\mathcal{D}}_{5,R}$, the decoupled state keeps track of the traces from the origin state 2, so none of the two is pruned, since they are not dominated by any state $s^{\mathcal{D}}_{i,R}$ (the root node $s^{\mathcal{D}}_{1,A}$ of **NestedDFS** is not yet visited).

As indicated before, in our emptiness checking algorithm we will use split decoupled states only within **NestedDFS**. The seed state $s^{\mathcal{D}}_A$ of **NestedDFS** will always be the l^A_G-successor of an accepting state $s^{\mathcal{D}}$ backtracked from in **DFS**. Every member state of $s^{\mathcal{D}}_A$ is accepting, or can be reached with L_I-transitions from an accepting state.

4.3 Putting Things Together: Decoupled NDFS

We are now ready to describe our adaptation of the standard NDFS algorithm to decoupled compositions. The pseudo-code is shown in Figure 7. The differences w.r.t the standard algorithm (Figure 2) are highlighted in blue. The basic structure of the algorithm is preserved. It starts by putting the decoupled initial state $s^{\mathcal{D}}_0$ onto the Stack in **CheckEmptiness**, and launches the main **DFS** from it.

In **DFS**, the control flow does not change, decoupled states are generated in depth-first order by recursion, updating the stack accordingly. There are however three differences to the standard variant:

1. Before generating the successors, we call **CheckLocalAccept** on each accepting decoupled state $s^{\mathcal{D}}$. This detects cycles resulting from L_I-transitions, i.e., cycles that occur "within" a decoupled state. To this end, we check whether there exists a component $\mathcal{A}^i$ for which an accepting local state s^i_a is reached that can reach itself using only internal labels L^i_I (the set of such local states can be precomputed, so that the check becomes a lookup operation). We can then construct an accepting run for the composed system by appending the L^i_I-cycle to the sequence of states that reaches s^i_a in $s^{\mathcal{D}}$ for $\mathcal{A}^i$. Note that it suffices if a single component moves and all other components remain in a reached accepting state.
2. Instead of doing duplicate checking, the algorithm performs dominance pruning, pruning a new decoupled state $t^{\mathcal{D}}$ if all its member states have been reached in an already visited decoupled state $r^{\mathcal{D}}$.

CheckEmptiness($\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$):
  Stack $\leftarrow \langle s_0^{\mathcal{D}} \rangle$
  $V \leftarrow \emptyset$
  $V' \leftarrow \emptyset$
  DFS($s_0^{\mathcal{D}}$)
  return empty

NestedDFS($s_R^{\mathcal{D}}$):
  for all $t_R^{\mathcal{D}}$ s.t. $s_R^{\mathcal{D}} \rightarrow_{\mathcal{D}} t_R^{\mathcal{D}}$ **do**
    if $\exists r_R^{\mathcal{D}} \in V'$ s.t. $t_R^{\mathcal{D}} \subseteq_R r_R^{\mathcal{D}}$ **then**
      continue
    if $\forall \mathcal{A}^i : \exists s : s \in t_R^{\mathcal{D}}[\mathcal{A}^i]_s$ **then**
      return cycle
    if $\exists r^{\mathcal{D}} \in$ Stack s.t. $r^{\mathcal{D}} \subseteq t_R^{\mathcal{D}}$ **then**
      return cycle
    $V' = V' \cup \{t_R^{\mathcal{D}}\}$
    NestedDFS($t_R^{\mathcal{D}}$)

DFS($s^{\mathcal{D}}$):
  $V = V \cup \{s^{\mathcal{D}}\}$
  if $A^{\mathcal{D}}(s^{\mathcal{D}})$ **then**
    CheckLocalAccept($s^{\mathcal{D}}$)
  for all $t^{\mathcal{D}}$ s.t. $s^{\mathcal{D}} \rightarrow_{\mathcal{D}} t^{\mathcal{D}}$ **do**
    if $\exists r^{\mathcal{D}} \in V$ s.t. $t^{\mathcal{D}} \subseteq r^{\mathcal{D}}$ **then**
      continue
    push(Stack, $t^{\mathcal{D}}$)
    DFS($t^{\mathcal{D}}$)
    pop(Stack)
  if $A^{\mathcal{D}}(s^{\mathcal{D}})$ **then**
    Let $s_A^{\mathcal{D}}$ s.t. $s^{\mathcal{D}} \xrightarrow{l_G^A}_{\mathcal{D}} s_A^{\mathcal{D}}$.
    NestedDFS($s_A^{\mathcal{D}}$)
    $V' = V' \cup \{s_A^{\mathcal{D}}\}$

CheckLocalAccept($s^{\mathcal{D}}$):
  if $\exists \mathcal{A}^i, s \in s^{\mathcal{D}}[\mathcal{A}^i] : A^i(s) \wedge s \xrightarrow{l_I \in L_I^i}{}^{+} s$
  then return cycle

Fig. 7. Adaptation of a standard NestedDFS for lasso search in decoupled compositions of NBA.

3. As discussed in Section 4.2, when we launch **NestedDFS** at a decoupled state $s^{\mathcal{D}}$, we do so on the acceptance-split l_G^A-successor $s_A^{\mathcal{D}}$ of $s^{\mathcal{D}}$.

NestedDFS now starts in the acceptance-split $s_A^{\mathcal{D}}$, and traverses split transitions as per Definition 5. On generation of a new state $t_R^{\mathcal{D}}$, we perform dominance pruning against the decoupled states visited during all prior calls to **NestedDFS**. If in an $t_R^{\mathcal{D}}$ for every component $\mathcal{A}^i$ there exists a reference state $s \in S^i$ that is reachable from itself, so $s \in t^{\mathcal{D}}[\mathcal{A}^i]_s$, then we can construct a cycle. As we will show in Theorem 2, this test is guaranteed to find all cycles that start from an accepting state $\boldsymbol{s}_A \in s_A^{\mathcal{D}}$.

Note that we cannot check for a non-empty intersection with states $r^{\mathcal{D}}$ on *Stack*, since these are not split relative to the reference states of $s_A^{\mathcal{D}}$. Thus, since we do not know from which local state in $r^{\mathcal{D}}$ the state in the intersection was reached, such a non-empty intersection would *not* imply a cycle. What we can do, however, is check for dominance instead, as an algorithm optimization inspired by [21]. The pseudo-code in Figure 7 does so by checking whether $t_R^{\mathcal{D}} \supseteq r^{\mathcal{D}}$, where the $\supseteq$ relation between a split vs. non-split state is simply evaluated based on the overall sets $t_R^{\mathcal{D}}[\mathcal{A}^i]$ vs. $r^{\mathcal{D}}[\mathcal{A}^i]$ of reached components states. If this domination relation holds true, then the reachability issue mentioned in the previous section is resolved because *all* $\boldsymbol{t} \in r^{\mathcal{D}}$ are then reachable from $s_A^{\mathcal{D}}$ – including those $\boldsymbol{t}$ from which an accepting state $\boldsymbol{s} \in s_A^{\mathcal{D}}$ is reachable. Lemma 1 in the next section will spell out this argument as part of our correctness proof.

Observe that splitting a decoupled state incurs an increase in the size of the state representation, as the same local state may be reached from several reference states. More importantly, as dominance pruning is weaker on split states (which after all is the purpose of the split operation) the size of the search space may increase. As shown

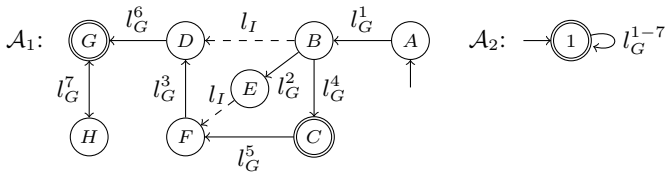


Fig. 8. Illustration of the component NBAs used in Example 1.

by the example in Figure 5, though, there is no easy way around the splitting, since the algorithm has to be able to know *from which* component state the successors states are reached. Assuming a component has M accepting states, then in the worst case all local successor states that are shared between these accepting states can be visited M times across all **NestedDFS** invocations. Unless some of the decoupled states revisiting the same member state are pruned by dominance pruning, it can actually happen that the revisits multiply across the components, so the size of the decoupled state space in **NestedDFS** can potentially be exponentially larger than the standard state space. As we shall see in our experimental evaluation, typically such blow-ups do not seem to occur.

In case we want to construct a lasso, we need to store a pointer to the predecessor of each decoupled state and the label of the generating transition. With this, we can, for each component $\mathcal{A}^i$ separately, reconstruct a trace $\boldsymbol{\pi}$ of a state $\boldsymbol{t} \in t^{\mathcal{D}}$ reached from a state $\boldsymbol{s} \in s^{\mathcal{D}}$ where $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}}) = \pi^G(\boldsymbol{\pi})$. Here, for a decoupled state $t^{\mathcal{D}}$ that was reached from another decoupled state $s^{\mathcal{D}}$, by $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}})$ we denote the global trace via which $t^{\mathcal{D}}$ was reached from $s^{\mathcal{D}}$. This can be done in time polynomial in the size of the component and linear in the length of $\pi^G(s^{\mathcal{D}}, t^{\mathcal{D}})$. Since the traces for all components are synchronized via $\pi^G(\boldsymbol{\pi})$, we add the required internal labels for each component in between every pair of global labels. We remark that, to decide if a lasso exists, we do not need to store any predecessor or generating label pointers.

We next show on an example how our algorithm works.

Example 1. The model has two component NBAs $\mathcal{A}_1, \mathcal{A}_2$ illustrated in Figure 8. It is a variant of an example from [25]. The Figure should be self-explanatory, we remark that all global transitions $l_G^1, \ldots l_G^7$ induce a self loop in the only state 1 of $\mathcal{A}_2$.

CheckEmptiness starts by putting $s_0^{\mathcal{D}}$ onto *Stack* and enters **DFS**$(s_0^{\mathcal{D}})$. Let $s_1^{\mathcal{D}} = \langle\{B, D\}, \{1\}\rangle$, $s_2^{\mathcal{D}} = \langle\{E, F\}, \{1\}\rangle$, and $s_3^{\mathcal{D}} = \langle\{D\}, \{1\}\rangle$ be the successors generated along the trace l_G^1, l_G^2, l_G^3 in **DFS**. Since $s_3^{\mathcal{D}} \subseteq s_1^{\mathcal{D}} \in V$, $s_3^{\mathcal{D}}$ is pruned and the search backtracks to $s_1^{\mathcal{D}}$. Say **DFS** selects the transition via l_G^4 next, generating the state $s_4^{\mathcal{D}} = \langle\{C\}, \{1\}\rangle$ and its l_G^5-successor $s_5^{\mathcal{D}} = \langle\{F\}, \{1\}\rangle$. Then $s_5^{\mathcal{D}}$ is pruned because it is dominated by $s_2^{\mathcal{D}} \in V$, and the search backtracks from $s_4^{\mathcal{D}}$, which is accepting.

Thus, **NestedDFS**$(s_{5,A}^{\mathcal{D}})$ is invoked, where $s_{5,A}^{\mathcal{D}} = \langle\langle\{C\}_C\rangle, \langle\{1\}_1\rangle\rangle$, because C and 1 are accepting local states that become the reference states of $s_{5,A}^{\mathcal{D}}$. **NestedDFS** will follow the trace $l_G^5, l_G^3, l_G^6, l_G^7, l_G^7$, which among others generates the state $s_{6,R}^{\mathcal{D}} = \langle\langle\{G\}_C\rangle, \langle\{1\}_1\rangle\rangle$ by l_G^6, and ends in $s_{7,R}^{\mathcal{D}} = \langle\langle\{G\}_C\rangle, \langle\{1\}_1\rangle\rangle$. The latter is pruned, because it is dominated by $s_{6,R}^{\mathcal{D}}$, which is contained in V'. No cycle is reported. This is correct, because the only member state $(C, 1)$ of $s_{5,A}^{\mathcal{D}}$ does not occur on a cycle.

Fig. 9. Illustration of the exponential separations to ample sets (left) and unfolding (right).

DFS then backtracks to $s_1^{\mathcal{D}} = \langle\{B, D\}, \{1\}\rangle$ and generates its remaining successor $s_8^{\mathcal{D}} = \langle\{G\}, \{1\}\rangle$ via l_G^6. **DFS** further generates the l_G^7-successors of $s_8^{\mathcal{D}}$ and eventually backtracks from $s_8^{\mathcal{D}}$, invoking **NestedDFS**$(s_{8,A}^{\mathcal{D}})$, where $s_{8,A}^{\mathcal{D}} = \langle\langle\{G\}_G\rangle, \langle\{1\}_1\rangle\rangle$.

After two transitions via l_G^7 the resulting state $s_{9,R}^{\mathcal{D}} = \langle\langle\{G\}_G\rangle, \langle\{1\}_1\rangle\rangle$ satisfies the condition that for all components $\mathcal{A}_i$ $\exists s : s \in s_{9,R}^{\mathcal{D}}[\mathcal{A}^i]_s$, namely G and 1. Thus, a cycle is reported. It is induced by the trace $l_G^1, l_I, l_G^6, l_G^7, l_G^7$.

Note that no decoupled state in the second **NestedDFS** is pruned, since none of them is dominated by a state in V' of the first **NestedDFS** invocation. In particular, $s_{8,A}^{\mathcal{D}} = \langle\{G\}_G, \{1\}_1\rangle$ is not dominated by $s_{6,R}^{\mathcal{D}} = \langle\{G\}_C, \{1\}_1\rangle$, because the reference states differ – G and 1 for $s_{8,R}^{\mathcal{D}}$ and C and 1 for $s_{6,R}^{\mathcal{D}}$.

4.4 Relation to other State-Space Reduction Methods

The comparison to ample set pruning and Petri-net unfolding from Section 3.3 carries over directly to liveness checking via simple adaptations to the examples, see Figure **??**.

Theorem 2. ***CheckEmptiness*** *with explicit-state search and ample sets pruning is exponentially separated from* ***CheckEmptiness*** *with decoupled search.*

Proof (sketch). The argument from Section 3.3 remains valid. With the states B_i accepting (see Figure **??**, left), explicit-state search with ample sets pruning in the worst case has to exhaust the entire state space. It invokes **NestedDFS** on the accepting state $(B_i)^n$ and, worst-case, needs to exhaust the state space again to detect the cycle. Decoupled search invokes **NestedDFS** on the initial state restricted to the component states B_i. Every successor of that state closes the cycle via an arbitrary l_G^b transition. So there are only three decoupled states overall (including the acceptance-restricted initial state). □

Theorem 3. *Constructing a complete unfolding prefix is exponentially separated from* ***CheckEmptiness*** *with decoupled search.*

Proof (sketch). The component states B_i are made accepting and internal transitions $B_i \to A_i$ are added to the model (see Figure **??**, right). Unfolding constructs a complete prefix as described in Section 3.3, plus one event for each new internal transition.[4] Decoupled search generates the two states as described. The second state has $\{A_i, B_i, C_i\}$ reached for all components, its successor via l_G is pruned. **NestedDFS** is invoked on its restriction to B_i, in which all A_i get reached via the new internal transitions. The l_G-successor of this state closes the cycle, so there are only four decoupled states. □

[4] A weaker cut-off rule is required for liveness checking that can only increase the prefix size [8].

5 Decoupled NDFS Correctness

We now show the correctness of our approach. In Lemmas 1, 2, 3, we show that if our algorithm reports a cycle, then there exists an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$. In Theorem 2, we then show that decoupled NDFS does not miss an accepting run.

We first show that the optimization of checking dominance of states in **NestedDFS** against states on the stack is sound, i.e., that an accepting run exists.

Lemma 1. *Let $r^{\mathcal{D}}$ be a decoupled state on the current **DFS** Stack, and let $t_R^{\mathcal{D}}$ be a decoupled state generated by **NestedDFS**. If $t_R^{\mathcal{D}} \supseteq r^{\mathcal{D}}$, then there exists an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$.*

Proof. Let $s^{\mathcal{D}}$ be the accepting state that is backtracked from in **DFS**, i.e., the current **NestedDFS** was invoked on its l_G^A-successor $s_A^{\mathcal{D}}$.

From Theorem 1 we know that if $s_2^{\mathcal{D}}$ is reachable from $s_1^{\mathcal{D}}$, then for every state $\boldsymbol{s}_2 \in s_2^{\mathcal{D}}$ there exists a state $\boldsymbol{s}_1 \in s_1^{\mathcal{D}}$ such that $\boldsymbol{s}_1 \xrightarrow{\boldsymbol{\pi}} \boldsymbol{s}_2$, where $\pi^G(\boldsymbol{\pi}) = \pi^G(s_1^{\mathcal{D}}, s_2^{\mathcal{D}})$.

This result also holds for decoupled states reached in **NestedDFS** from states in **DFS**. This is because the acceptance-split transition l_G^A only restricts the set of reached member states of $s^{\mathcal{D}}$ in $s_A^{\mathcal{D}}$, so in particular $s_A^{\mathcal{D}} \subseteq s^{\mathcal{D}}$. Furthermore, split transitions generating states behind $s_A^{\mathcal{D}}$ do not affect reachability of member states of these split-decoupled states compared to their non-split counterparts.

In particular, (1) for every state $\boldsymbol{s}_2 \in s^{\mathcal{D}}$ there exists a state $\boldsymbol{s}_1 \in s_0^{\mathcal{D}}$ that reaches $\boldsymbol{s}_2$ on a trace $\boldsymbol{\pi}$ where $\pi^G(\boldsymbol{\pi}) = \pi^G(s_0^{\mathcal{D}}, s^{\mathcal{D}})$, which, with $s_A^{\mathcal{D}} \subseteq s^{\mathcal{D}}$ also holds for all $\boldsymbol{s}_2 \in s_A^{\mathcal{D}}$; and (2) for every state $\boldsymbol{t} \in t_R^{\mathcal{D}}$ there exists an accepting state $\boldsymbol{s}_A \in s_A^{\mathcal{D}}$ that reaches $\boldsymbol{t}$ on a trace $\boldsymbol{\pi}$ where $\pi^G(\boldsymbol{\pi}) = \pi^G(s_A^{\mathcal{D}}, t_R^{\mathcal{D}})$.

Since $r^{\mathcal{D}}$ is on *Stack*, it holds that every $\boldsymbol{s} \in s_A^{\mathcal{D}}$ is reachable from a $\boldsymbol{r} \in r^{\mathcal{D}}$, and, with $t_R^{\mathcal{D}} \supseteq r^{\mathcal{D}}$, that every $\boldsymbol{r} \in r^{\mathcal{D}}$ is reachable from an accepting state $\boldsymbol{s}_A \in s_A^{\mathcal{D}}$.

Let $pred(s_1^{\mathcal{D}}, s_2^{\mathcal{D}}, \boldsymbol{s}_2)$ be a function that, if $s_2^{\mathcal{D}}$ is reachable from $s_1^{\mathcal{D}}$ and $\boldsymbol{s}_2 \in s_2^{\mathcal{D}}$, outputs a state $\boldsymbol{s}_1 \in s_1^{\mathcal{D}}$ that reaches $\boldsymbol{s}_2$ via a trace $\boldsymbol{\pi}$ with $\pi^G(s_1^{\mathcal{D}}, s_2^{\mathcal{D}}) = \pi^G(\boldsymbol{\pi})$.

Let $\boldsymbol{s}_0$ be a state reached in both $t_R^{\mathcal{D}}$ and $r^{\mathcal{D}}$, and let $\boldsymbol{s}_1 = pred(r^{\mathcal{D}}, t_R^{\mathcal{D}}, \boldsymbol{s}_0)$ be its predecessor in $r^{\mathcal{D}}$. If $\boldsymbol{s}_1 = \boldsymbol{s}_0$, then we are done, because there exists a lasso $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_0, \ldots, \boldsymbol{s}_A, \ldots, \boldsymbol{s}_0, \ldots, \boldsymbol{s}_A$, where $\boldsymbol{s}_A$ is an accepting state traversed in $s_A^{\mathcal{D}}$. Such an accepting state exists because all member states of a decoupled state in **NestedDFS** are reachable from an accepting state in $s_A^{\mathcal{D}}$.

If $\boldsymbol{s}_1 \neq \boldsymbol{s}_0$, then we iterate and set $\boldsymbol{s}_i = pred(r^{\mathcal{D}}, t_R^{\mathcal{D}}, \boldsymbol{s}_{i-1})$, where such $\boldsymbol{s}_i$ exist because $r^{\mathcal{D}} \subseteq t_R^{\mathcal{D}}$. Because there are only finitely many states in $r^{\mathcal{D}}$, eventually we get $\boldsymbol{s}_i = \boldsymbol{s}_j$ (where $j < i$) and there exists a lasso as follows:

First, there exists a cycle $\boldsymbol{s}_i, \ldots, \boldsymbol{s}_{i-1}, \ldots, \boldsymbol{s}_j = \boldsymbol{s}_i$, where between every pair of states $\boldsymbol{s}_k, \boldsymbol{s}_{k-1}$ an accepting state $\boldsymbol{s}_{k,A}$ in $s_A^{\mathcal{D}}$ is traversed, for the same reason as before. We can obviously shift and truncate the cycle to start right after and end in $\boldsymbol{s}_{i,A}$. The prefix of the lasso is $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{i,A}$. □

Lemmas 2 and 3 show the soundness of our main termination criterion, and of **CheckLocalAccept**.

Lemma 2. *Let $t_R^{\mathcal{D}}$ be a split decoupled state generated in **NestedDFS**. If for every component $\mathcal{A}^i$ there exists a component state s^i such that $s^i \in t^{\mathcal{D}}[\mathcal{A}^i]_{s^i}$, then there exists an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$.*

Proof. Let $s_A^{\mathcal{D}}$ be the acceptance-split decoupled state from which **NestedDFS** was started. If for every component $\mathcal{A}^i$ such an s^i exists, then the state $\boldsymbol{s} = (s^1, \ldots, s^n)$ is reachable in both $s_A^{\mathcal{D}}$ and $t_R^{\mathcal{D}}$. By the construction of the reached state sets $t_R^{\mathcal{D}}[\mathcal{A}^i]_{s^i}$, $\boldsymbol{s}$ is reachable from itself and is accepting. Hence, there exists a lasso $\boldsymbol{s}_0, \ldots, \boldsymbol{s}, \ldots, \boldsymbol{s}$. □

Lemma 3. *Let $t^{\mathcal{D}}$ be an accepting decoupled state generated in **DFS** such that a cycle is reported by **CheckLocalAccept**($t^{\mathcal{D}}$), then an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ exists.*

Proof. By prerequisite, there exists an accepting member state $\boldsymbol{s}$ of $t^{\mathcal{D}}$. If **CheckLocalAccept**($t^{\mathcal{D}}$) reports a cycle, then there exists a component $\mathcal{A}^i$, where an accepting state $s^i \in t^{\mathcal{D}}[\mathcal{A}^i]$ is reached that lies on an cycle induced by transitions labelled with L_I^i. Thus, we can set the local state of $\mathcal{A}^i$ in $\boldsymbol{s}$ to s^i, and the lasso looks as follows: $\boldsymbol{s}_0, \ldots, \boldsymbol{s}, \ldots, \boldsymbol{s}$, where on the cycle only $\mathcal{A}^i$ moves. □

We are now ready to prove the correctness of our decoupled NDFS algorithm.

Theorem 4. *Let $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ be the composition of n NBA and let $\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$ be its decoupled composition. Then **CheckEmptiness**($\mathcal{A}^1 \parallel_{\mathcal{D}} \ldots \parallel_{\mathcal{D}} \mathcal{A}^n$) reports a cycle if and only if an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ exists.*

Proof. If **CheckEmptiness** reports a cycle, then by Lemmas 1, 2, and 3, which cover exactly the cases where a cycle is reported, an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$ exists.

For the other direction, assume that $\boldsymbol{\rho}$ is an accepting run for $\mathcal{A}^1 \parallel \ldots \parallel \mathcal{A}^n$. Let $\boldsymbol{s}_a$, with $0 \leq a < k$, be the accepting state that starts the cycle of the lasso $\boldsymbol{\rho}_p = \boldsymbol{s}_0, \ldots, \boldsymbol{s}_a$, $\boldsymbol{\rho}_c = \boldsymbol{s}_{a+1}, \ldots, \boldsymbol{s}_k$, where $\boldsymbol{s}_a = \boldsymbol{s}_k$. Let $\boldsymbol{\pi} = l_1, \ldots, l_k$ be the trace on which $\boldsymbol{s}_k$ is reached, i.e., for all $1 \leq i < k : \langle \boldsymbol{s}_i, l_{i+1}, \boldsymbol{s}_{i+1} \rangle \in \rightarrow$.

By Theorem 1, there exists a decoupled state $s^{\mathcal{D}}$ reached in **DFS** that contains $\boldsymbol{s}_a$.

If $\boldsymbol{\pi}$ is such that for all $a < i \leq k : l_i \in L_I$, i.e., the cycle $\boldsymbol{\rho}_c$ is induced only by internal labels, we next proof that **CheckLocalAccept**($s^{\mathcal{D}}$) reports a cycle: As $\boldsymbol{s}_a$ is accepting, $s^{\mathcal{D}}$ is accepting, too, so unless a cycle is reported before, eventually **CheckLocalAccept**($s^{\mathcal{D}}$) is called. If $\boldsymbol{\rho}_c$ is induced by only internal labels, then, because there cannot be any component interaction via L_I-transitions, there must exist a component $\mathcal{A}^i$ for which the local state s^i in $\boldsymbol{s}_a$ reaches itself with only L_I^i-transitions. We can pick any such $\mathcal{A}^i$ and ignore transitions from $\boldsymbol{\rho}_c$ that are labelled by an element of $L_I \setminus L_I^i$, since these are not required for an accepting cycle. Consequently, **CheckLocalAccept**($s^{\mathcal{D}}$) reports a cycle.

We next show that, if $\boldsymbol{\pi}$ contains a global label on the cycle, i.e., there exists an $i \in \{a+1, \ldots, k\}$ such that $l_i \in L_G$, then, unless a cycle is reported before, **NestedDFS**($s_A^{\mathcal{D}}$) reports a cycle, where $s_A^{\mathcal{D}}$ is the l_G^A-successor of $s^{\mathcal{D}}$.

Assume for contradiction that this is not the case, i.e., no cycle has been reported before, and **NestedDFS**($s_A^{\mathcal{D}}$) does not report a cycle. Let **NestedDFS**($s_A^{\mathcal{D}}$) be the first call to **NestedDFS** that misses a cycle, although an $\boldsymbol{s}_a \in s_A^{\mathcal{D}}$ that is on a cycle exists.

If $\boldsymbol{s}_a$ is on a cycle, then by Theorem 1 there exists a decoupled state $t^{\mathcal{D}}$ reachable from $s_A^{\mathcal{D}}$ that also contains $\boldsymbol{s}_a$. The result of Theorem 1 holds in this case because, by the definition of split transitions, the splitting does not affect reachability of member states. So there exists $t_R^{\mathcal{D}}$ reachable from $s_A^{\mathcal{D}}$ that contains $\boldsymbol{s}_a$.

Dining Philosophers										Ring Topology									
	SPIN			Cunf			DecNDFS				SPIN			Cunf			DecNDFS		
#$\mathcal{A}$	Time	#States	Mem	Time	#E	M	Time	#States	M	#$\mathcal{A}$	Time	#States	Mem	Time	#E	M	Time	#S	M
3	0.0	76	129	0.00	75	6	0.00	36	8	6	0.10	81.5K	133	0.00	342	6	0.00	8	8
4	0.0	348	129	0.00	162	6	0.00	97	8	7	0.95	560K	157	0.00	484	7	0.00	9	8
5	0.0	2000	129	0.00	293	6	0.00	272	8	8	8.35	3.7M	303	0.01	651	7	0.00	10	8
6	0.0	9416	131	0.01	482	7	0.01	783	8	9	73.6	24.6M	1367	0.01	843	8	0.00	11	8
7	0.2	45132	139	0.01	735	8	0.06	2290	8	10	-	-	-	0.01	1060	9	0.00	12	8
8	1.3	212K	175	0.02	1066	9	0.60	6761	8	15	-	-	-	0.03	2525	17	0.00	17	8
9	7.9	992K	333	0.02	1481	11	5.49	20.1K	9	20	-	-	-	0.10	4570	37	0.00	22	8
10	46.8	4.6M	993	0.04	1994	15	56.7	59.9K	14	25	-	-	-	0.22	7240	74	0.01	27	8
11	278.0	21.6M	3965	0.04	2386	18	558	179K	44	50	-	-	-	3.80	30K	917	0.06	52	8
12	-	-	-	0.06	2874	23	-	-	-	75	-	-	-	-	-	-	0.26	77	8

Fig. 10. Statistics on the two scaling models, where #$\mathcal{A}$ is the number of philosophers, resp. the number of NBAs, Time is runtime in seconds, #States (#S) and #E are the number of visited states, resp. generated events, and Mem (M) is the memory usage in MiB.

Denote by $\boldsymbol{\pi}_c = l_{a+1}, \dots l_k$ the cycle part of $\boldsymbol{\pi}$. Because $\boldsymbol{s}_a$ is an accepting member state of $s^{\mathcal{D}}$, all its component states s^i_A become reference states in $s^{\mathcal{D}}_A$. Therefore, assuming that $\pi^G(s^{\mathcal{D}}_A, t^{\mathcal{D}}_R) = \pi^G(\boldsymbol{\pi}_c)$, for all components we have $s^i_A \in t^{\mathcal{D}}_R[\mathcal{A}^i]_{s^i_A}$ and a cycle is reported. If this is not the case, then either (1) $s^{\mathcal{D}}$ was not reached in **DFS**, or (2) $t^{\mathcal{D}}_R$ was not reached in **NestedDFS**$(s^{\mathcal{D}}_A)$.

In case (1), there must exist a state $s^{\mathcal{D}}_P \supseteq s^{\mathcal{D}}$ that prunes $s^{\mathcal{D}}$. But then, $s^{\mathcal{D}}_P$ contains $\boldsymbol{s}_a$, too, and **NestedDFS** was called on its l^A_G-successor $s^{\mathcal{D}}_{P,A}$ and the cycle of $\boldsymbol{s}_a$ was missed before, in contradiction.

For (2), either (a) there exists a state $t^{\mathcal{D}}_{P,R} \supseteq_R t^{\mathcal{D}}_R$ that was reached in a prior invocation of **NestedDFS** on an accepting state $s^{\mathcal{D}}_{P,A}$, or (b) a state $t^{\mathcal{D}}_{P,R} \supseteq t^{\mathcal{D}}_R$ was reached in **NestedDFS**$(s^{\mathcal{D}}_A)$ before $t^{\mathcal{D}}_R$. In both cases, $t^{\mathcal{D}}_R$ is pruned and the cycle through $\boldsymbol{s}_a$ is missed. Case (a) can only happen if $s^{\mathcal{D}}_{P,A}$ contains $\boldsymbol{s}_a$, too, because the reference states of $s^{\mathcal{D}}_A$ need to be a subset of the ones of $s^{\mathcal{D}}_{P,A}$. But then, the cycle of $\boldsymbol{s}_a$ was missed before, in contradiction. For (b), if $t^{\mathcal{D}}_R \subseteq_R t^{\mathcal{D}}_{P,R}$, then for all $\mathcal{A}^i$ we have $s^i_A \in t^{\mathcal{D}}_{P,R}[\mathcal{A}^i]_{s^i_A}$, so the cycle would have been reported before, in contradiction.

The reachability argument in (1,2a,2b) applies recursively to all predecessors of $s^{\mathcal{D}}$ in **DFS**, and of $t^{\mathcal{D}}_R$ in **NestedDFS**$(s^{\mathcal{D}}_A)$, so, unless a cycle is reported before, eventually a state $s^{\mathcal{D}}$ is reached in **DFS** that contains $\boldsymbol{s}_a$, and a state $t^{\mathcal{D}}_R$ with $s^i_A \in t^{\mathcal{D}}_R[\mathcal{A}^i]_{s^i_A}$ in **NestedDFS**$(s^{\mathcal{D}}_A)$. □

6 Experimental Evaluation

We implemented a prototype of the decoupled NDFS algorithm from Figure 7. The input is specified in the Hanoi Omega-Automata format [1], describing a set of NBAs synchronized via global labels as in Definition 2. We compare our prototype to the SPIN model checker [19] (v6.5.1), and to the Cunf Petri-net unfolding tool [29] (v1.6.1). We also experimented with the symbolic model checkers NuSMV and PRISM [3, 24], but both are significantly outperformed by the other methods. We conjecture that this is because both systems are not specifically designed for asynchronous execution of processes, or LTL model checking. For SPIN, we translate each NBA to a process where

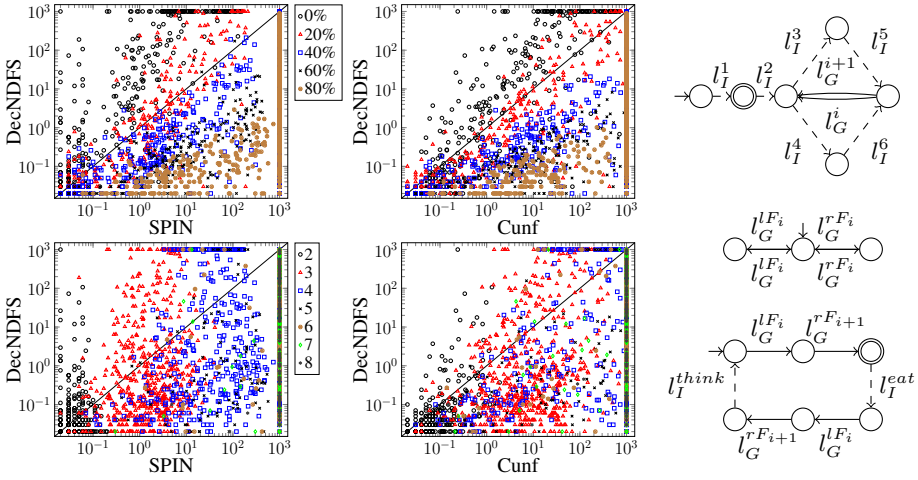


Fig. 11. Left part: scatterplots with the runtime of DecNDFS on the y-axis and the one of SPIN (left column) and Cunf (right column) on the x-axis, on randomly generated models. Each point represents one instance. In the top row, we highlight different ratios of local labels with different colors/shapes, in the bottom row we highlight different numbers of components. Right part: illustrations of the ring model (top) and the fork (middle) and philosopher (bottom) NBAs of the philosophers model. Initial (accepting) states are marked by an incoming arrow (double circle).

NBA states are represented by state labels, internal transitions by `goto` statements, and global transitions by rendezvous channel operations. For the latter, SPIN only supports synchronization of two processes at a time, so we restrict the models to global transitions with exactly two components. We model acceptance for SPIN explicitly using a monitor process that gets into an accepting state if all processes are in a local accepting state. The translation for Cunf encodes NBA states as net places and transitions as net transitions into a single Petri net, ignoring the individual components. In our prototype and in SPIN, when a lasso is reported or the algorithm proved that no lasso exists within the cut-off limits, we say that the instance was *solved*. For Cunf, we attempt to construct a complete unfolding prefix. We consider an instance solved if the construction terminates, i.e., we do not actually check the liveness property. The experiments were performed on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 15 min (4 GiB). Our code and models are publicly available [**?**].

We compare SPIN with standard options, i.e., with partial-order reduction enabled, Cunf with the cut-off rule of [10], and decoupled search (DecNDFS), using two kinds of benchmarks: (1) two scaling examples to showcase the behaviour on well-known models. One is an encoding of the dining philosophers problem, the other is a ring-shaped synchronisation topology. Both are illustrated in Figure 10 (right). The philosophers model has $2N$ NBAs, N philosophers and N forks, synchronized by global transitions $l_G^{lF_i}$ and $l_G^{rF_i}$. After synchronizing with its left and right fork, a philosopher can perform an internal *eat* transition; after releasing the forks it can perform an internal *think* transition. In the ring-topology model, each component can enter a diamond-shaped region

Ratio	#	SPIN	Cunf	DecNDFS	#$\mathcal{A}$	#	SPIN	Cunf	DecNDFS
					2	750	721	**750**	749
					3	750	696	**745**	712
0%	1050	**373**	369	319	4	750	411	243	**541**
20%	1050	385	384	**462**	5	750	130	114	**372**
40%	1050	397	384	**573**	6	750	49	53	**266**
60%	1050	422	394	**723**	7	750	24	34	**180**
80%	1050	468	431	**888**	8	750	14	23	**145**
$\sum$	5250	2045	1962	**2965**	$\sum$	5250	2045	1962	**2965**

Fig. 12. Number of solved instances on the random models as a function of the ratio of internal transitions (left) and the number of components #$\mathcal{A}$ (right).

via internal transitions, followed by a synchronization with its left or right neighbor via l_G^i or l_G^{i+1}. No accepting run exists for either model. Moreover, (2) we use a set of random automata, where for each combination of a ratio of internal transitions in $\{0\%, 20\%, \ldots, 80\%\}$, i.e., the number of transitions labelled with L_I divided by the total number of transitions, and a number of components in $\{2, \ldots, 8\}$, we generated sets of 150 random graphs. Each component has 15 to 100 local states, out of which up to 3% are accepting (at least one). We ensure that none of the instances has an internal accepting cycle to focus on more interesting cases. One could easily implement a lookup similar to **CheckLocalAccept**, which is necessary for DecNDFS, for the other methods, too, which then essentially simplifies the problem to basic reachability.

In Figure 9, we show detailed statistics for the scaling models, with increasing number of components $\#\mathcal{A}$ (Time in seconds, #States is the sum of states visited in both DFSs, #E is the number of events in the prefix, Memory in MiB). In dining philosophers, SPIN and DecNDFS show similar results. SPIN has a runtime advantage in the larger instances of roughly a factor of 2, but DecNDFS uses only a fraction of the memory. Cunf clearly outperforms both. This model is not very well suited to decoupled search. Only half of the NBAs have internal transitions, and only two each, and there are no non-deterministic transitions that DecNDFS could represent compactly. On the ring-topology model, SPIN manages to exhaust the search space for up to 9 components. Cunf and DecNDFS scale significantly higher, the number of decoupled states grows only linearly in the number of components. Cunf on the other hand does show a blow-up and runs out of memory between 50 and 75 components. This showcase example only serves to illustrate a near-to-optimal case for decoupled search reductions, which likely does not carry over in this extent to real-world models.

In Figure 10 (left part), we show detailed runtime behaviour in terms of scatter plots with a per-instance comparison on the random models. Each point corresponds to one instance, where the x-value is the runtime of SPIN, resp. Cunf, and the y-value is the runtime of DecNDFS, so points below the diagonal indicate an advantage of DecNDFS. Different ratios of internal labels (top row) and numbers of components (bottom row) are depicted in different colors/shapes. We observe that, as expected, with a higher ratio of internal transitions, the advantage of DecNDFS increases significantly. For all ratios, DecNDFS clearly improves with a higher number of components.

In Figure 11, for the same benchmark set we show the number of solved instances as a function of the ratio (left) and of the number of components (right). Here, we see that from around 20% internal transitions, DecNDFS consistently beats both SPIN

and Cunf. SPIN and Cunf also benefit from the decrease in synchronizing statements, although not as much as DecNDFS. On the right, we see that starting with 4 component NBAs ($\#\mathcal{A}$), DecNDFS consistently beats SPIN and Cunf. While SPIN and Cunf show a significant decline with more components, this effect is less pronounced for DecNDFS.

7 Concluding Remarks and Future Work

We have presented an approach to adapt decoupled search, an AI planning technique to mitigate the state-space explosion, to the verification of liveness properties of composed NBAs. Specifically, we have adapted a standard on-the-fly algorithm for checking ω-regular properties, nested depth-first search (NDFS), and proven its correctness. The necessary adaptations essentially pertain to the conditions that identify the existence of accepting runs, which must be handled differently given the different properties of decoupled states. Our approach extends the scope of decoupled search from safety properties, as done in [12], to liveness properties. Our experimental evaluation has shown that decoupled search can yield significant reductions in search effort across random models that consist of a set of synchronized NBAs, and simple scaling showcase examples.

We have focused on a verification problem for composed NBAs that is sufficiently general to cover significant cases like automata-based LTL model checking. We believe that our solution can be adapted to other verification problems for composed NBAs, including Büchi automata with multiple acceptance conditions such as *generalized Büchi automata*, and language intersection of the involved automata. Indeed, NDFS has successfully been used for emptiness checking of generalized NBA. We are confident that decoupled NDFS can be adapted to the compilation introduced by [32], where an additional "counter component" is added to keep track of the components that already have an accepting cyle during the nested DFS. Concretely, we believe that the verification problem of generalized NBA can be handled with adaptations by our approach: In the compilation by [32], the counter component increases its local state from 1 to n (assuming n components), one by one whenever component i has an accepting state. We can essentially apply the same compilation in decoupled NDFS, restricting the set of local states of $\mathcal{A}^i$ to the accepting ones when the counter is increased from i to $i+1$ by a separate acceptance-split transition $l_G^{A^i}$ for each $\mathcal{A}_i$. This ensures that a global cycle includes an accepting state for all components.

There are several interesting topics for future work, like the adaptation of optimizations proposed for basic NDFS (e.g. [21, 31]), or the combination with orthogonal state space reduction methods, as previously done in the context of AI planning for partial-order reduction [15], symmetry reduction [17], and symbolic search [16]. Having focused on NDFS [5, 21, 31] in this work, we believe that the adaptation of SCC-based algorithms is a promising line of research [6, 11], extending the scope of decoupled search further to model checking of CTL properties [23].

Acknowledgments. We thank Álvaro Torralba for helpful discussions about the state-splitting approach. Daniel Gnad was supported by the German Research Foundation (DFG), as part of project grant HO 2169/6-2, "Star-Topology Decoupled State Space Search". Jörg Hoffmann's research group has received support by DFG grant 389792660 as part of TRR 248 (see `perspicuous-computing.science`).

References

1. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kretínský, J., Müller, D., Parker, D., Strejcek, J.: The hanoi omega-automata format. Lecture Notes in Computer Science, vol. 9206, pp. 479–486. Springer (2015)
2. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (1986)
3. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th International Conference, CAV 2002, Copenhagen, Denmark, July 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer (2002)
4. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (2001)
5. Courcoubetis, C., Vardi, M.Y., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal Methods in System Design **1**(2/3), 275–288 (1992). https://doi.org/10.1007/BF00121128
6. Couvreur, J.: On-the-fly verification of linear temporal logic. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20-24, 1999, Proceedings, Volume I. Lecture Notes in Computer Science, vol. 1708, pp. 253–271. Springer (1999). https://doi.org/10.1007/3-540-48119-2_16
7. Emerson, E.A., Sistla, A.P.: Symmetry and model-checking. Formal Methods in System Design **9**(1/2), 105–131 (1996)
8. Esparza, J., Heljanko, K.: A new unfolding approach to LTL model checking. Lecture Notes in Computer Science, vol. 1853, pp. 475–486. Springer (2000). https://doi.org/10.1007/3-540-45022-X
9. Esparza, J., Heljanko, K.: Implementing LTL model checking with net unfoldings. In: Vardi, M.Y., Dwyer, M.B., Chechik, M. (eds.) Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN-01). pp. 37–56. Springer-Verlag, Toronto, Canada (May 2001). https://doi.org/10.1007/3-540-45139-0_4
10. Esparza, J., Römer, S., Vogler, W.: An improvement of mcmillan's unfolding algorithm. Formal Methods in System Design **20**(3), 285–310 (2002)
11. Geldenhuys, J., Valmari, A.: Tarjan's algorithm makes on-the-fly LTL verification more efficient. In: Jensen, K., Podelski, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2988, pp. 205–219. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_18
12. Gnad, D., Dubbert, P., Lluch-Lafuente, A., Hoffmann, J.: Star-topology decoupling in SPIN. In: del Mar Gallardo, M., Merino, P. (eds.) Proceedings of the 25th International Symposium on Model Checking of Software (SPIN'18). Lecture Notes in Computer Science, Springer (2018)
13. Gnad, D., Hoffmann, J.: Star-topology decoupled state space search. Artificial Intelligence **257**, 24 – 60 (2018)
14. Gnad, D., Hoffmann, J.: On the relation between star-topology decoupling and petri net unfolding. In: Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19). AAAI Press (2019)
15. Gnad, D., Hoffmann, J., Wehrle, M.: Strong stubborn set pruning for star-topology decoupled state space search. Journal of Artificial Intelligence Research **65**, 343–392 (2019). https://doi.org/10.1613/jair.1.11576

16. Gnad, D., Torralba, Á., Hoffmann, J.: Symbolic leaf representation in decoupled search. In: Fukunaga, A., Kishimoto, A. (eds.) Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17). AAAI Press (2017)
17. Gnad, D., Torralba, Á., Shleyfman, A., Hoffmann, J.: Symmetry breaking in star-topology decoupled search. In: Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17). AAAI Press (2017)
18. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem, Lecture Notes in Computer Science, vol. 1032. Springer (1996)
19. Holzmann, G.: The Spin Model Checker - Primer and Reference Manual. Addison-Wesley (2004)
20. Holzmann, G.J., Peled, D.A.: An improvement in formal verification. In: Hogrefe, D., Leue, S. (eds.) Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques, Berne, Switzerland, 1994. IFIP Conference Proceedings, vol. 6, pp. 197–211. Chapman & Hall (1994)
21. Holzmann, G.J., Peled, D.A., Yannakakis, M.: On nested depth first search. In: Grégoire, J., Holzmann, G.J., Peled, D.A. (eds.) The Spin Verification System, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, August, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 32, pp. 23–31. DIMACS/AMS (1996). https://doi.org/10.1090/dimacs/032/03
22. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods in System Design **9**(1/2), 41–75 (1996). https://doi.org/10.1007/BF00625968
23. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM **47**(2), 312–360 (2000). https://doi.org/10.1145/333979.333987
24. Kwiatkowska, M.Z., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proceedings of the 23rd International on Conference Computer Aided Verification (CAV'11). Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011)
25. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 968–983. Springer (2013)
26. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: von Bochmann, G., Probst, D.K. (eds.) Proceedings of the 4th International Workshop on Computer Aided Verification (CAV'92). pp. 164–177. Lecture Notes in Computer Science, Springer (1992)
27. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
28. Peled, D.A.: Combining partial order reductions with on-the-fly model-checking. Formal Methods in System Design **8**(1), 39–64 (1996). https://doi.org/10.1007/BF00121262
29. Rodríguez, C., Schwoon, S.: Cunf: A tool for unfolding and verifying petri nets with read arcs. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13). pp. 492–495 (2013)
30. Roggenbach, M.: Determinization of büchi-automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]. Lecture Notes in Computer Science, vol. 2500, pp. 43–60. Springer (2001)
31. Schwoon, S., Esparza, J.: A note on on-the-fly verification algorithms. Lecture Notes in Computer Science, vol. 3440, pp. 174–190. Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_12

32. Tauriainen, H.: Nested emptiness search for generalized büchi automata. Fundamenta Informaticae **70**(1-2), 127–154 (2006)
33. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design **1**(4), 297–322 (1992)