Star-Topology Decoupling in SPIN

Gnad, Daniel; Dubbert, Patrick; Lluch Lafuente, Alberto; Hoffmann, Jorg

Published in:
SPIN 2018: Model Checking Software

Link to article, DOI:
10.1007/978-3-319-94111-0_6

Publication date:
2018

Document Version
Peer reviewed version

Citation (APA):
Star-Topology Decoupling in SPIN

Daniel Gnad\(^1\), Patrick Dubbert\(^1\), Alberto Lluch Lafuente\(^2\), and Jörg Hoffmann\(^1\)

\(^1\) Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
\{gnad, hoffmann\}@cs.uni-saarland.de, patrick@dexpot.de

\(^2\) Technical University of Denmark, Copenhagen, Denmark
albl@dtu.dk

Abstract. Star-topology decoupling is a state space search method recently introduced in AI Planning. It decomposes the input model into components whose interaction structure has a star shape. The decoupled search algorithm enumerates transition paths only for the center component, maintaining the leaf-component state space separately for each leaf. This is a form of partial-order reduction, avoiding interleavings across leaf components. It can, and often does, have exponential advantages over stubborn set pruning and unfolding. AI Planning relates closely to model checking of safety properties, so the question arises whether decoupled search can be successful in model checking as well. We introduce a first implementation of star-topology decoupling in SPIN, where the center maintains global variables while the leaves maintain local ones. Preliminary results on several case studies attest to the potential of the approach.

1 Introduction

AI Planning develops algorithms that, given an initial state \(s_0\) (an assignment to a vector of state variables), a goal formula \(G\), and a set \(A\) of actions (transition rules), find an action sequence that transforms \(s_0\) into a state \(s\) s.t. \(s \models G\). In other words, AI Planning addresses reachability checking in compactly described transition systems. This relates closely to model checking of safety properties, a well-known connection (e.g. \[3, 6, 27–29\]) that has been exploited to transfer techniques. In the context of the SPIN model checker \[24\], AI Planning heuristic search methods have been adapted to SPIN \[9, 10\], and compilations from Promela to AI Planning languages have been designed \[8\].

Here we adapt a new method from AI Planning, star-topology decoupling \[14, 15\], to model checking. Contrary to other methods developed in AI, which typically aim at finding solution paths quickly, the major strength of star-topology decoupling lies in proving unreachability: in a model checking setting, verifying correctness of safety properties. We provide a first implementation in SPIN, and initial empirical results.

Star-topology decoupling decomposes the input problem into components identified by a partition of state variables. Two components interact if there is an action reading or updating state variables from both of them. Star-topology decoupling chooses components whose interactions take a star shape, where there is a center component to which all interactions are incident. All other components are then referred to as leaves. Given such a topology, the leaves depend only indirectly on each other, via the center. The decoupled search algorithm exploits this through a two-level search, where only the
center is considered at the primary level, while each leaf is considered separately at the secondary level. Multiplication of states across leaf components is avoided.

Star-topology decoupling relates to partial-order reduction (e.g. [37, 32, 19, 12, 34]), in that it avoids interleavings of leaf paths. It can be viewed as a variant of unfolding, exploiting star shapes by organizing the unfolding in terms of transition paths over the center, which ensures by design that there are no cross-leaf conflicts. Star-topology decoupling can have exponential advantages over other partial-order reduction methods. Consider the following excerpt of Gnad and Hoffmann’s [15] results:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Exp</th>
<th>SSS</th>
<th>Unf</th>
<th>STD</th>
<th>Exp</th>
<th>SSS</th>
<th>Unf</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevators</td>
<td>100</td>
<td>21</td>
<td>17</td>
<td>3</td>
<td>41</td>
<td>1,941.81</td>
<td>1,941.5</td>
<td>543.3</td>
</tr>
<tr>
<td>Logistics</td>
<td>63</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>27</td>
<td>1,121.2</td>
<td>1,121.2</td>
<td>118.4</td>
</tr>
<tr>
<td>Micronic</td>
<td>150</td>
<td>50</td>
<td>45</td>
<td>30</td>
<td>145</td>
<td>154.6</td>
<td>152.3</td>
<td>143.1</td>
</tr>
<tr>
<td>NoMystery</td>
<td>40</td>
<td>11</td>
<td>11</td>
<td>7</td>
<td>40</td>
<td>266.2</td>
<td>248.8</td>
<td>101.3</td>
</tr>
<tr>
<td>TPP</td>
<td>30</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>11</td>
<td>192.5</td>
<td>192.5</td>
<td>12.4</td>
</tr>
<tr>
<td>Woodworking</td>
<td>100</td>
<td>11</td>
<td>20</td>
<td>22</td>
<td>16</td>
<td>109,174.4</td>
<td>199.9</td>
<td>1.2</td>
</tr>
<tr>
<td>∑ (over all)</td>
<td>1144</td>
<td>202</td>
<td>196</td>
<td>123</td>
<td>435</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Left: #state spaces successfully exhausted in 30 minutes/4 GB memory. Right: State-space representation size (#integer variables, in thousands, used in the final representation). Exp: explicit-state search without enhancements. SSS: strong stubborn sets (as per [39]). Unf: unfolding (using Cunf [34] given the presence of read arcs). STD: star-topology decoupling.

Here we observe that, in automata networks such as described in Promela, decoupled search can be applied by viewing “local” transitions, affecting only a single process \( P \), as being part of a leaf component \( P \); while viewing non-local transitions, affecting more than one process, as being part of the center component. In the simplest case, where processes communicate only via global variables, this takes the global variables as the center and takes the local variables of each process as a leaf. But also more general forms of communication, via channels, can be viewed in this way. The decoupled search then explores non-local transitions at the primary level, and local transitions at the secondary level. We supply initial empirical evidence suggesting that this form of decomposition can be useful in the verification of safety properties.

## 2 Star-Topology Decoupling

We first describe star-topology decoupling in the context of AI Planning where it was invented. We give a brief outline and refer to Gnad and Hoffmann [15] for details. In Section 2.2, we prove correctness of star-topology decoupling for reachability checking. Finally, we show complementarity to previous state-space reduction methods.

### 2.1 Decoupling in AI planning

An AI Planning task is a tuple \((V, A, s_0, G)\). \( V \) is a finite set of state variables \( v \), each with a finite domain \( D_v \). A state \( s \) is an assignment to \( V \) and \( s_0 \) is the initial state. The goal \( G \) is a partial assignment to \( V \), interpreted as a conjunctive formula where
The planning problem is to decide whether there exists not. The applicability and outcome is defined by
\[ s \text{ to applicable} \]
leaf states that remain reachable in the successor decoupled state (i.e., are compatible with
\[ \text{used to illustrate that a new leaf state } a \text{ becomes reachable via leaf action } l. \]
Dashed lines indicate leaf states that remain reachable in the successor decoupled state (i.e., are compatible with \( a^+ \)).

\[ s \models (v, d) \text{ iff } s(v) = d. A \text{ is a set of actions, each action } a \text{ associated with two partial assignments to } V \text{ namely the precondition } \text{pre}[a] \text{ and effect } \text{eff}[a]. \]
An action is applicable to } s \text{ if } s \models \text{pre}[a]. \text{ If so, the outcome state of applying } a \text{ in } s, \text{ denoted } s[a], \text{ is defined by } s[a](v) = \text{eff}[a](v) \text{ where } \text{eff}[a] \text{ is defined, and } s[a](v) = s(v) \text{ where not. The applicability and outcome } s[\pi] \text{ of an action sequence } \pi \text{ is defined accordingly. The planning problem is to decide whether there exists } \pi \text{ such that } s_0[\pi] \models G. \]

As an example, simple yet enough to show exponential separations from previous methods, say that \( V = \{t, p_1, \ldots, p_n\} \) where \( t \) encodes the position of a truck on a map with two locations \( l, r \) and each \( p_i \) encodes the position of a package. We have \( D_l = \{l, r\} \) and \( D_{p_i} = \{l, r, T\} \) where \( T \) stands for being in the truck. In \( s_0 \), all variables have value \( l \). The goal is to bring all packages to \( r \), i.e., \( G = \{(p_1, r), \ldots, (p_n, r)\} \).

The actions drive, e.g., \text{drive}lr with precondition \( \{(t, l)\} \) and effect \( \{(t, r)\} \); or load a package, e.g., \text{loadp}l with precondition \( \{(t, l), (p_1, l)\} \) and effect \( \{(p_1, T)\} \); or unload a package, e.g., \text{unloadp}r with precondition \( \{(t, r), (p_1, T)\} \) and effect \( \{(p_1, r)\} \). The decoupled state space of the example, which we define next, is illustrated in Figure 2.

Let \( P = \{P_1, P_2, \ldots\} \) be a partitioning of \( V \), i.e., \( \biguplus_{P_i \in P} P_i = V \). Consider the undirected graph with vertices \( P \) and an arc \( (P_1, P_2) \) for \( P_1 \neq P_2 \) if there exists \( a \in A \) s.t. the set \( V_a \) of variables touched by \( a \) (defined in either \( \text{pre}[a] \) or \( \text{eff}[a] \)) intersects both \( P_1 \) and \( P_2 \). We say that \( P \) is a star-topology decomposition if there exists a unique \( C \in P \) s.t. all arcs in the graph are incident on \( C \). In that case, \( C \) is the center and all other \( L \in P \) are leaves. In the example, \( P = \{\{t\}, \{p_1\}, \ldots, \{p_n\}\} \) is a star-topology decomposition with center \( C = \{t\} \) and leaves \( L_i = \{p_i\} \).

Refer to value assignments to \( C \) as center states \( s^C \), and to value assignments to a leaf \( L \) as leaf states \( s^L \). These are the atomic composites of the search graph built by decoupled search. The search starts with the center state \( s^C_0 := s_0|_C \). It then augments \( s^C_0 \) with a full exploration of leaf states reachable given \( s^C_0 \); it iteratively applies all leaf actions \( a^L \), affecting only some \( L \), where \( s^L \models \text{pre}[a^L]|_C \). Denote the set of all \( s^L \) reached this way by \( S^L[s^C_0] \). Then \( s^L_0 \) together with \( S^L[s^C_0] \) forms a decoupled state. In the example, \( s^C_0 = \{(t, l)\} \) and \( S^L[s^C_0] = \{(p_i, l), (p_i, T) | 1 \leq i \leq n\} \). Observe that, given the star-topology decomposition, the leaves do not interact with each other, so
any combination of leaf states \( s^{L_1, \ldots, L_n} \in S^L[C_0] \) is jointly reachable. Intuitively, fixing the center, the leaves – which interact only via the center – become independent.

Given a decoupled state \((s^C, S^L[C^C])\), the successor center states \(r^C\) are those reached from \(s^C\) by some center action \(a^C\), affecting \(C\), that is applicable: \(s^C \models \text{pre}[a^C]|_C\), and for every leaf \(L\) there exists \(s^L \in S^L[C^C]\) s.t. \(s^L \models \text{pre}[a^C]|_L\). Each such \(r^C\) reached by \(a^C\) is added to the search graph. Then \(r^C\) is augmented into a decoupled state by 1) selecting from \(S^L[C^C]\) the subset \(S^L[C^C, a^C]\) of leaf states compatible with \(a^C\), and 2) setting \(S^L[r^C]\) to be all leaf states reachable from \(r^C\) and \(S^L[C^C, a^C]\).

The goal \(G\) is reached if, for some decoupled state \((s^C, S^L[C^C])\) in the search graph, \(s^C \models G|_C\) and for every leaf \(L\) there exists \(s^L \in S^L[C^C]\) s.t. \(s^L \models G|_L\).

In the example, the only successor center state of \((s^C_0, S^L[C^C_0])\) is \(s^C_1 = \{(t, r)\}\) reached by the center action \(a^C = \text{drive}lr\). We get \(S^L[s^C_1] = \{(p_i, l), (p_i, T), (p_i, r)\}\) \(1 \leq i \leq n\), because 1) all \(s^L \in S^L[C^C_0]\) are compatible with \(\text{drive}lr\), and 2) given \(s^C\) we can unload each package at \(r\). Thus the goal is reached in the decoupled state \((s^C_1, S^L[s^C_1])\). The complete decoupled state space \(S\) of the task contains a third decoupled state \((s^C_2, S^L[s^C_2])\) that results from applying \(\text{drive}l\) in \((s^C_1, S^L[s^C_1])\).

### 2.2 Correctness of decoupling for reachability analysis

Given the star-topology decomposition, any decoupled state \((s^C, S^L[C^C])\) generated this way represents exactly the states \(s\) reachable in the original task using the same center-action sequence \(\pi^C\) that led to \((s^C, S^L[C^C])\). Those \(s\) are exactly the ones where \(s|_C = s^C\) and, for every leaf \(L\), \(s|_L \in S^L[C^C]\). In particular, the goal is reachable in decoupled search iff it is reachable in the original task. When the goal is reached in \((s^C, S^L[C^C])\), a solution can be extracted by backchaining from the leaf states \(G|_L = S^L[C^C]\). Duplicate decoupled states can be pruned, so exploring the set of reachable decoupled states in any order leads to a finite decoupled search space \(S\). In consequence, star-topology decoupling guarantees correctness for reachability checking:

**Theorem 1.** Star-topology decoupling captures reachability exactly, i.e., a state \(s\) is reachable in a task iff there exists a decoupled state in \(S\) in which \(s\) is represented. A path to \(s\) can be extracted in time linear in \(|S|\). Checking if a conjunctive property \(c\) is reachable in \(S\) is linear in the number of decoupled states.

**Proof (sketch).** Every decoupled state \((s^C, S^L[C^C])\) captures exactly those states that are reachable via the center-action subsequence \(\pi^C\) that led to \((s^C, S^L[C^C])\). Thus, all states reachable in the task are represented by a decoupled states in \(S\). A path to an \(s\) in \((s^C, S^L[C^C])\) can efficiently be obtained by augmenting \(\pi^C\) with leaf action sequences for each leaf component by backchaining from the leaf states that compose \(s\). Reachability of a conjunctive property \(c\) in a decoupled state is done by checking, for each partition \(P\) separately, if the projection of \(c\) onto \(P\) is reached.

The full proofs are available in Gnäd and Hoffmann [15].
2.3 Complementarity from other methods

Several other methods relate to star-topology decoupling in that they can also lead to exponential reductions in search effort. Decoupling can still be exponentially more efficient, as our transportation example shows:

There are exactly three reachable decoupled states: the initial state \((s_0^C, S^L[s_0^C])\), its successor \((s_1^C, S^L[s_1^C])\) from driverl, plus the only successor of \((s_1^C, S^L[s_1^C])\), reached via driverl (which differs from \((s_0^C, S^L[s_0^C])\) because \((p_i, r)\) is reached for each \(p_i\)).

The separation into partial component states may be reminiscent of abstraction methods, which over-approximate the set of reachable states given one such component (e.g. [4, 7, 22, 23]). Star-topology decoupling is very different, partitioning the variables to avoid enumerating combinations of independent leaf component assignments. Given this particular structure, star-topology decoupling captures reachability exactly.

The search space under strong stubborn set (SSS) pruning has size exponential in the number of packages. This is because an SSS on the initial state must include a load\(p_i\)l action to make progress to the goal, must include drive\(r\) as that interferes with load\(p_i\)l, and must then include all other load\(p_j\)l actions as these interfere with drive\(r\). So all subsets of packages that may be loaded at \(l\) are enumerated.

In unfolding, the non-consumed preconditions of load actions on the truck position induce read arcs. Both ways of encoding these (consuming and producing the truck position, or place replication) result in an unfolding enumerating all subsets of loaded packages. In contextual Petri nets that support read arcs natively [2], the same explosion arises in the enumeration of “event histories”.

Symmetry breaking (e.g. [35, 11, 13, 33, 5]) is complementary to star-topology decoupling, simply because the leaf partitions do not need to be symmetric. In our example, where the leaves are symmetric, even perfect symmetry breaking keeps track of the number of packages at every location, enumerating all possible combinations.

3 Implementation in SPIN

We implemented star-topology decoupling in the most recent version of SPIN (6.4.7). We focus on reachability properties only (more general properties are a topic for future work). Our current implementation is preliminary in that it does not handle the full Promela language accepted by SPIN itself. We specify the handled fragment below. Let us first describe the star-topology decomposition and the modified search algorithms.

In Promela models, a star-topology decomposition arises directly from the formulation as interacting processes. Each process becomes a leaf component on its own; but anything that affects more than a single process is grouped into the center component. Concretely, each of the statements \(st\) in a process type \(t\) corresponds to either a local transition, affecting only local variables or advancing the process location; or a global transition, namely a channel operation, a statement that affects a global variable, or a run command invoking a new process. Then the instantiations of \(t\) can be made leaf processes if \(t\) contains at least one local transition. All remaining processes, global variables, as well as channels, together form the center component. This partitioning ensures that every interaction across processes involves the center component. Center
and leaf states are defined as assignments to the respective parts of the model, and center (resp. leaf) transitions are ones that affect the center (resp. only that leaf). The annotation of statements to become leaf or center transitions is done fully automatic.

Our implementation of decoupled search is minimally intrusive. We keep SPIN’s current state \( s^C \) to store the center state. Alongside \( s^C \), we maintain a data structure storing the associated set \( S^L[s^C] \) of reached leaf states. The decoupled search algorithm is then adopted as follows. The primary search only branches over center transitions, i.e., center processes and center transitions in leaf processes. We loop over \( S^L[s^C] \) to determine the center transitions enabled by the reached leaf states. A center transition \( t^C \) applied to a decoupled state \( (s^C, S^L[s^C]) \) can have updates on leaf processes, so we need to compute the set \( S^L[s^C, t^C] \) as before, and apply the leaf updates of \( t^C \) to the states in that set. Afterwards, \( S^L[s^C, t^C] \) is augmented by all reachable leaf states to obtain the successor decoupled state \( (r^C, S^L[r^C]) \). We perform duplicate checking over decoupled states, testing the center states first to save runtime.

The remaining issue with our implementation is SPIN’s parsing process. Due to the generation of model-specific code, the distinction between local (leaf) and global (center) transitions cannot be identified anymore within the verifier itself, but must be identified at Promela level. SPIN’s parsing process must be extended to identify the leaf-vs-center information, and to communicate that to the verifier. Currently, our implementation supports this for assignments, conditions, basic control constructs (do...od, if...fi), all unary and binary operators, channel operations (send/receive, both synchronous and buffered), and run commands. We do not yet support timeout, unless, and channel polling statements (empty/full...), nor the process constraints priority and provided, nor more complex constructs like c-code and inline. For atomic and d_step sequences, we handle basic compounds of statements, series of conditions, assignments, and channel operations, but not more complex control flows.

Regarding the relation to other search methods used in SPIN, partial-order reduction is orthogonal, and potentially exponentially worse, as we have already shown in the planning context. The same is true of statement merging, which can only reduce the number of states local to a process, merging statements that only touch local variables. It cannot merge statements that have conditions on global variables, and thus cannot tackle the exponential search space size in our transportation example. Similar arguments apply to reduction methods based on \( \tau \)-confluence (e.g., [21, 20]). Note also that leaf transitions, while local to a process, may be relevant to the property being checked (e.g., be part of a conjunctive reachability property as in planning).

4 Experiments

We performed experiments on several case studies, selected to suit the Promela fragment we can currently handle, and selected to showcase the potential of star-topology decoupling. We emphasize that the experiments are preliminary and we do not wish to make broad claims regarding their significance. Our implementation and all models used in the below are available at https://bitbucket.org/dagnad/decoupled-spin-public.

We run the scalable variant of Peterson’s Mutex algorithm from Lynch [31], an elevator control model developed by Armin Biere and used as benchmark in several
papers (e.g. [9, 10]), the X.509 protocol from Jøsang [26], and a client-server communication protocol. The latter is a toy example we created for the purpose of this study, as a simple pattern to highlight the kind of structure relevant to star-topology decoupling. The model consists of a server process handling requests from a scalable number of client processes. Communication is via two channels. In star-topology decoupling the clients become leaf components, and the technique is beneficial if there is local content within each client. To show this, we experiment with two variants, EmptyC where the clients do nothing other than communicating with the server, and NonEmptyC where each client increments a local variable from 0 to 1. For illustrative purposes, we also include the transportation planning example described earlier. Modeling this in Promela is straightforward. We scale the number of packages from 1 to 50.

We compare our decoupled-search SPIN (STD) to SPIN 6.4.7 with standard settings, providing no additional command line options (SPIN), and to a configuration disabling statement merging (-M) and partial-order reduction (-POR). All configurations exhaust the entire state space, using the verifier options -A -E. Restricting ourselves to safety properties, we removed any never claims from the models. We use runtime (memory) limits of 60 min (32 GB). Figure 3 shows the results, scaling each case study until all configurations run out of memory (indicated by a “-”).

STD works very well in Peterson, significantly reducing memory consumption and runtime. To a lesser extent, STD also has advantages in Elevator and X.509. In Client-Server, as expected STD is beneficial only if there is local content in the clients. In the transportation case study adopted from planning, STD excels. This is not a relevant observation in model checking per se, but points to the power star-topology decoupling may in principle have over previous search methods in SPIN.

The number of decoupled states is consistently smaller than the number of states in SPIN (and, e.g., by 2 orders of magnitude in Peterson). Where the reduction is relatively small, it is outweighed by the runtime overhead of handling decoupled states. Regarding the search depth, keep in mind that the maximum depth of STD is that of the center transitions only. The depth bound can, thus, in general be kept significantly smaller for STD, leading to a reduced memory consumption for the search stack.

5 Conclusion

Star-topology decoupling is a novel approach to reduce state space size in checking reachability properties. Our implementation in SPIN is still preliminary, but exhibits encouraging performance on some case studies. As work in the planning domain has already shown, star-topology decoupling is orthogonal to, and may have exponential advantages over, partial-order reduction, symmetry breaking, symbolic representations, and heuristic search. It can also be fruitfully combined with all of these [18, 17, 16, 15].

We believe that the technique’s application to model checking is promising, and we hope that our preliminary study will have an impact in this direction. Foremost, more realistic case studies are required. Client-server architectures, and concurrent programs under weak memory constraints (e.g. [25, 30, 36, 1]), carry promise insofar as such models might exhibit relevant local structure to be exploited in leaf components:
**Fig. 3.** Performance of SPIN with default options (SPIN), disabling statement merging (-M) and partial-order reduction (-POR), and with star-topology decoupling (STD). We show runtime (in seconds) and memory consumption (in GB), as well as the number of stored states (#S), and the maximum search depth (D) reported by SPIN. Best runtime/memory is highlighted in **bold face**.

client/process parts that may read the server/shared memory state, and that may have indirect effects thereupon, but that do not update it directly.

An important research challenge is the extension to liveness properties. This can be approached by encoding the property that is to be checked as a finite automaton that is added to the center component. We can then perform a lasso search in the decoupled state space. This should work similar as in explicit state search, yet potentially leads to more interaction between center and leaves, which has a negative influence on the performance of star-topology decoupling. Further research topics include the combination with other search techniques like (lossy) state compression, and application to other model checking frameworks.

**Acknowledgments.** Daniel Gnad was partially supported by the German Research Foundation (DFG), as part of project grant HO 2169/6-1, "Star-Topology Decoupled State Space Search".
References


