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
[2] Technical University of Denmark, Copenhagen, Denmark

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, 4, 20–22]) that has been exploited to transfer techniques. In the context of the SPIN model checker [17], AI Planning heuristic search methods have been adapted to SPIN [6, 7], and compilations from Promela to AI Planning languages have been designed [5].

Here we adapt a new method from AI Planning, *star-topology decoupling* [9, 10], 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. [28, 25, 14, 8, 26]), 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 [10] results:

Benchmark	#	Exp	SSS	Unf	STD	Exp	SSS	Unf	STD
Elevators	100	21	17	3	**41**	1,941.8	1,941.5	543.3	**36.3**
Logistics	63	12	12	11	**27**	1,121.2	1,121.2	118.4	**12.1**
Miconic	150	50	45	30	**145**	154.6	152.3	143.1	**.7**
NoMystery	40	11	11	7	**40**	266.2	248.8	101.3	**3.9**
TPP	30	5	5	4	**11**	192.5	192.5	12.4	**.2**
Woodworking	100	11	20	**22**	16	109,174.4	199.9	**1.2**	4,274.2
$\sum$ (over all)	1144	202	196	123	**435**				

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 [29]). Unf: unfolding (using Cunf [26] 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 [10] for details.

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. s_0 is the *initial state*. The goal G is a partial assignment to V, interpreted as a conjunctive formula where $s \models (v, d)$ iff $s(v) = d$. A is a set of *actions*, each action a associated with two partial assignments to V namely the *precondition* $pre[a]$ and *effect* $\mathit{eff}[a]$. An action is *applicable* to s if $s \models pre[a]$. If so, the outcome state of applying a in s, denoted $s[\![a]\!]$, is defined by $s[\![a]\!](v) = \mathit{eff}[a](v)$ where $\mathit{eff}[a]$ is defined, and $s[\![a]\!](v) = s(v)$ where not. The applicability and outcome $s[\![\pi]\!]$ of an action sequence π is defined accordingly. The planning problem is to decide whether there exists π 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 $D_t = \{l, r\}$ 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. $drivelr$ with precondition $\{(t, l)\}$ and effect $\{(t, r)\}$; or load a package, e.g. $loadp_1l$ with precondition $\{(t, l), (p_1, l)\}$ and effect $\{(p_1, T)\}$; or unload a package, e.g. $unloadp_1r$ with precondition $\{(t, r), (p_1, T)\}$ and effect $\{(p_1, r)\}$.

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 $pre[a]$ or $\mathit{eff}[a]$) 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 $s_0^C := s_0|_C$. It then augments s_0^C with a full exploration of leaf states reachable given s_0^C: it iteratively applies all *leaf actions* a^L, affecting only some L, where $s_0^C \models pre[a^L]|_C$. Denote the set of all s^L reached this way by $S^L[s_0^C]$. Then s_0^C together with $S^L[s_0^C]$ forms a *decoupled state*. In the example, $s_0^C = \{(t, l)\}$ and $S^L[s_0^C] = \{(p_i, l), (p_i, T) \mid 1 \leq i \leq n\}$. Observe that, given the star-topology decomposition, the leaves do not interact with each other, so any combination of leaf states $s^{L_1}, \ldots, s^{L_n} \in S^L[s_0^C]$ is jointly reachable. Intuitively, fixing the center, the leaves – which interact only via the center – become independent.

Given a decoupled state $(s^C, S^L[s^C])$, the successor center states r^C are those reached from s^C by some *center action* a^C, affecting C, that is applicable: $s^C \models pre[a^C]|_C$, and for every leaf L there exists $s^L \in S^L[s^C]$ s.t. $s^L \models pre[a^C]|_L$. 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 $S^L[s^C]$ the subset $S^L[s^C, a^C]$ of leaf states compatible with a^C, and 2) setting $S^L[r^C]$ to be all leaf states reachable from r^C and $S^L[s^C, a^C]$.

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

In the example, the only successor center state of $(s_0^C, S^L[s_0^C])$ is $r^C = \{(t, r)\}$ reached by the center action $a^C = drivelr$. We get $S^L[r^C] = \{(p_i, l), (p_i, T), (p_i, r) \mid 1 \leq i \leq n\}$, because 1) all $s^L \in S^L[s_0^C]$ comply with $drivelr$, and 2) given r^C we can unload each package at r. Thus the goal is reached in the decoupled state $(r^C, S^L[r^C])$.

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

In our example, there are exactly three reachable decoupled states: the initial state $(s_0^C, S^L[s_0^C])$ and its successor $(r^C, S^L[r^C])$ from $drivelr$; plus the only successor of $(r^C, S^L[r^C])$, resulting from $driverl$ (which differs from $(s_0^C, S^L[s_0^C])$ because (p_i, r) is reached for each p_i). In contrast, the search space under strong stubborn set (SSS) pruning, and under unfolding, has size exponential in the number of packages. For SSS, this is because an SSS on the initial state must include a $loadp_il$ action to make progress to the goal, must include $drivelr$ as that interferes with $loadp_il$, and must then include all other $loadp_jl$ actions as these interfere with $drivelr$. 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 [2], that support read arcs natively, the same explosion arises in the enumeration of "event histories".

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).

Our implementation of decoupled search is minimally intrusive. We keep SPIN's current state now to store the center state s^C. Alongside now, 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 τ-*confluence* (e.g. [16, 15]). 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 [24], an elevator control model developed by Armin Biere and used as benchmark in several papers (e.g. [6, 7]), the X.509 protocol from Jøsang [19], 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 1 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.

Model		SPIN -M -POR Time	Mem	#S	D	SPIN Time	Mem	#S	D	Star-topology decoupling (STD) Time	Mem	#S	D
Peterson	3	0.04	0.13	33434	6924	**0**	0.13	2999	615	**0**	0.13	274	120
	4	19	1.14	8886434	1703147	0.53	0.21	533083	165342	**0.1**	**0.13**	6698	1615
	5	-	-	-	-	124	10.08	76620358	25309679	**4.16**	**0.27**	153548	27392
	6	-	-	-	-	-	-	-	-	**157**	**4.79**	3503908	473228
Elevator	3	0.23	0.14	99057	4609	0.12	**0.13**	78284	4950	**0.06**	**0.13**	7081	590
	4	3.15	0.19	685169	29487	1.49	0.17	498676	30239	**0.42**	**0.16**	37095	1643
	5	15.5	0.44	3620470	28638	5.02	0.33	2354211	27634	**2.38**	**0.26**	115077	1630
	6	95.7	1.86	18813600	30818	26.7	1.13	10868993	29712	**7.35**	**0.65**	359163	1728
	7	676	10.31	97574250	32998	153	5.56	49636481	31790	**25.5**	**2.08**	1119285	1826
	8	-	-	-	-	782	26.62	224704000	33868	**95.5**	**7.51**	3483243	1924
	9	-	-	-	-	-	-	-	-	**360**	**27**	10825893	2022
X.509		1.58	0.18	403311	91	**0**	**0.13**	3054	57	**0**	**0.13**	1090	35
Client-	6	0.72	0.15	141312	16235	**0.08**	**0.13**	32296	6830	0.15	0.14	13128	755
Server-	7	4.84	0.21	745472	63236	**0.42**	**0.15**	143741	27442	0.74	0.19	51037	1607
EmptyC	8	31.3	0.59	3801088	263835	**2.05**	**0.24**	507967	104759	3.3	0.42	192464	3297
	9	199	2.83	18874368	1062399	**8.53**	**0.44**	2206702	370926	14.6	1.36	708597	6274
	10	1160	12.38	91750400	4252067	**35**	**1.66**	8140911	1277049	63.2	5.34	2558800	13414
	11	-	-	-	-	**149**	**4.45**	29856762	4335070	256	21.11	9093557	28150
	12	-	-	-	-	**609**	**21.06**	109424300	14937082	-	-	-	-
Client-	5	2.69	0.18	450560	65354	0.05	**0.13**	23614	6209	**0.04**	**0.13**	3245	324
Server-	6	30.2	0.73	4816896	518833	0.33	**0.15**	132210	32645	**0.21**	**0.15**	13128	755
NonEmptyC	7	416	7.6	49545216	4256969	2.12	0.27	708019	172048	**1.04**	**0.21**	51037	1607
	8	-	-	-	-	14	0.74	3813278	882008	**4.86**	**0.53**	192464	3297
	9	-	-	-	-	83.8	3.79	19384754	4254923	**20.4**	**1.87**	708597	6274
	10	-	-	-	-	480	22.14	95568530	19967819	**87.1**	**7.57**	2558800	13414
	11	-	-	-	-	-	-	-	-	**369**	**30.39**	9093557	28150
Transport-	4	0.22	0.13	112735	329	**0**	0.13	31018	311	**0**	0.13	18	8
Planning	5	3.85	0.19	1240092	978	0.36	0.14	237249	892	**0**	**0.13**	21	9
	6	47.8	0.95	13641019	2923	3.14	0.24	1815310	2698	**0**	**0.13**	24	10
	7	728	12.8	150051220	8756	29	1.07	13954478	6404	**0**	**0.13**	27	11
	8	-	-	-	-	269	8.03	107967020	19740	**0**	**0.13**	30	12
	50	-	-	-	-	-	-	-	-	**0.01**	**0.13**	156	54

Fig. 1. 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**.

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 [13, 12, 11, 10].

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. [18, 23, 27, 1]), carry promise insofar as such models might exhibit relevant local structure to be exploited in leaf components: 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. Further research challenges include extension to liveness properties, combination with other search techniques like (lossy) state compression, and application to other model checking frameworks.

References

1. Alrahman, Y.A., Andric, M., Beggiato, A., Lluch-Lafuente, A.: Can we efficiently check concurrent programs under relaxed memory models in maude? In: Revised Selected Papers of the 10th International Workshop on Rewriting Logic and Its Applications (WRLA'14). pp. 21–41 (2014)
2. Baldan, P., Bruni, A., Corradini, A., König, B., Rodríguez, C., Schwoon, S.: Efficient unfolding of contextual Petri nets. Theoretical Computer Science 449, 2–22 (2012)
3. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence 147(1–2), 35–84 (2003)
4. Dräger, K., Finkbeiner, B., Podelski, A.: Directed model checking with distance-preserving abstractions. In: Valmari, A. (ed.) Proceedings of the 13th International SPIN Workshop (SPIN 2006). Lecture Notes in Computer Science, vol. 3925, pp. 19–34. Springer-Verlag (2006)
5. Edelkamp, S.: Promela planning. In: Ball, T., Rajamani, S. (eds.) Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN-03). pp. 197–212. Springer-Verlag, Portland, OR (May 2003)
6. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit model checking with hsf-spin. 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. 57–79. Springer-Verlag, Toronto, Canada (May 2001)
7. Edelkamp, S., Lluch-Lafuente, A., Leue, S.: Directed explicit-state model checking in the validation of communication protocols. International Journal on Software Tools for Technology Transfer 5(2-3), 247–267 (2004)
8. Esparza, J., Römer, S., Vogler, W.: An improvement of mcmillan's unfolding algorithm. Formal Methods in System Design 20(3), 285–310 (2002)
9. Gnad, D., Hoffmann, J.: Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In: Brafman, R., Domshlak, C., Haslum, P., Zilberstein, S. (eds.) Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15). pp. 88–96. AAAI Press (2015)
10. Gnad, D., Hoffmann, J.: Star-topology decoupled state space search. Artificial Intelligence 257, 24 – 60 (2018)
11. 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)
12. 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)
13. Gnad, D., Wehrle, M., Hoffmann, J.: Decoupled strong stubborn sets. In: Kambhampati, S. (ed.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16). pp. 3110–3116. AAAI Press/IJCAI (2016)
14. 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)
15. Groote, J.F., van de Pol, J.: State space reduction using partial tau-confluence. In: Nielsen, M., Rovan, B. (eds.) Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1893, pp. 383–393. Springer (2000)
16. Groote, J.F., Sellink, M.P.A.: Confluence for process verification. In: Lee, I., Smolka, S.A. (eds.) CONCUR '95: Concurrency Theory, 6th International Conference, Philadelphia, PA,

USA, August 21-24, 1995, Proceedings. Lecture Notes in Computer Science, vol. 962, pp. 204–218. Springer (1995)
17. Holzmann, G.: The Spin Model Checker - Primer and Reference Manual. Addison-Wesley (2004)
18. Jonsson, B.: State-space exploration for concurrent algorithms under weak memory orderings. SIGARCH Computer Architecture News 36(5), 65–71 (2008)
19. Jøsang, A.: Security protocol verication using spin. In: The First SPIN Workshop, Montreal, Quebec, Canada (1995)
20. Kupferschmid, S., Hoffmann, J., Dierks, H., Behrmann, G.: Adapting an AI planning heuristic for directed model checking. In: Valmari, A. (ed.) Proceedings of the 13th International SPIN Workshop (SPIN 2006). Lecture Notes in Computer Science, vol. 3925, pp. 35–52. Springer-Verlag (2006)
21. Kupferschmid, S., Hoffmann, J., Dräger, K., Finkbeiner, B., Dierks, H., Podelski, A., Behrmann, G.: Uppaal/dmc – abstraction-based heuristics for directed model checking. In: Grumberg, O., Huth, M. (eds.) Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07). Springer-Verlag (2007)
22. Kupferschmid, S., Hoffmann, J., Larsen, K.G.: Fast directed model checking via Russian doll abstraction. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). pp. 203–217. Springer-Verlag (2008)
23. Linden, A., Wolper, P.: A verification-based approach to memory fence insertion in PSO memory systems. In: Piterman, N., Smolka, S.A. (eds.) Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13). pp. 339–353. Springer-Verlag (2013)
24. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
25. 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)
26. 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)
27. Travkin, O., Mütze, A., Wehrheim, H.: SPIN as a linearizability checker under weak memory models. In: Proceedings of the 9th International Haifa Verification Conference (HVC'13). pp. 311–326 (2013)
28. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design 1(4), 297–322 (1992)
29. Wehrle, M., Helmert, M.: Efficient stubborn sets: Generalized algorithms and selection strategies. In: Chien, S., Do, M., Fern, A., Ruml, W. (eds.) Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14). AAAI Press (2014)