AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Mind the Shapes: Abstraction Refinement via Topology Invariants

by
Jörg Bauer, Tobe Toben and Bernd Westphal

AVACS Technical Report No. 22
June 2007
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Copyright © June 2007 by the author(s)
Author(s) contact: Bernd Westphal (`Bernd.Westphal@Informatik.Uni-Oldenburg.DE`).

Mind the Shapes: Abstraction Refinement via Topology Invariants

Jörg Bauer[1], Tobe Toben[2] and Bernd Westphal[2]

[1] Technical University of Denmark, Kongens Lyngby, Denmark, `joba@imm.dtu.dk`*
[2] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany,
`{toben,westphal}@informatik.uni-oldenburg.de`

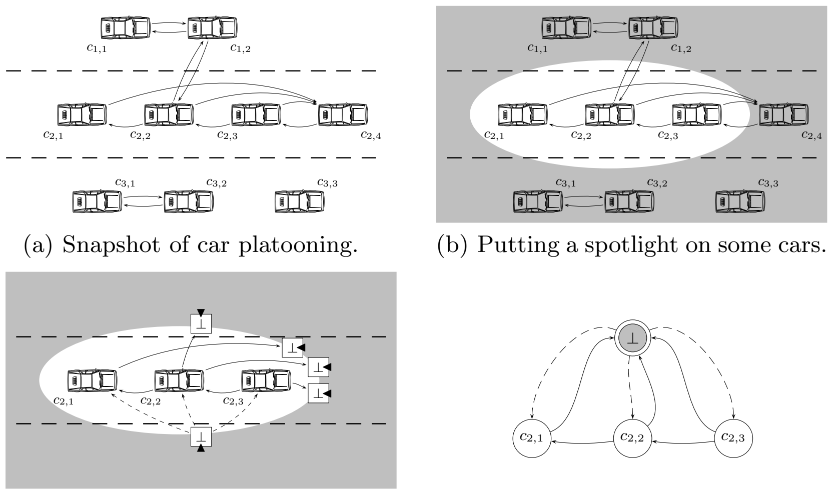
Abstract. Dynamic Communication Systems (DCS) are infinite state systems where an unbounded number of processes operate in an evolving communication topology. For automated verification of properties of DCS, finitary abstractions based on exploiting symmetry can be employed. However, these abstractions give rise to spurious behaviour that often inhibits to successfully prove relevant properties.
In this paper, we propose to combine a particular finitary abstraction with global system invariants obtained by abstract interpretation. These system invariants establish an over-approximation of possible communication topologies occurring at runtime, which can be used to identify and exclude spurious behaviour introduced by the finitary abstraction, which is thereby refined. Based on a running example of car platooning, we demonstrate that our approach allows to verify temporal DCS properties that no technique in isolation is able to prove.

1 Introduction

Formal verification of systems with dynamic process creation is an active research area. In [3], we characterised a certain class of such systems, the so-called Dynamic Communication Systems (DCS), by providing the formal description language *DCS protocols.* DCS protocols are complemented by Mett, a variant of temporal logic for requirements specification. In this work, we elaborate on an automated procedure for checking whether a DCS protocol satisfies a given Mett property. A manual procedure was briefly sketched in [3]. By bridging the technical gap between different analysis techniques with different strengths and weaknesses, we obtain a fully automated, integrated implementation, which benefits from synergetical effects.

Running Example. DCS are ubiquitous, most prominent among them mobile ad-hoc networks, service-oriented computing scenarios, or traffic control systems based on wireless communication. In order to demonstrate the appropriateness of our automated DCS verification technique we pick the characteristic real-world example *car platooning*, a traffic control system studied by the California PATH project [9]. In order to optimise traffic throughput on highways and reduce energy consumption, they propose that cars shall dynamically form platoons (cf.

* The author was partly sponsored by the project SENSORIA, IST-2005-016004.



(a) Snapshot of car platooning. (b) Putting a spotlight on some cars.

(c) Abstract from those in darkness. (d) Summarised process.

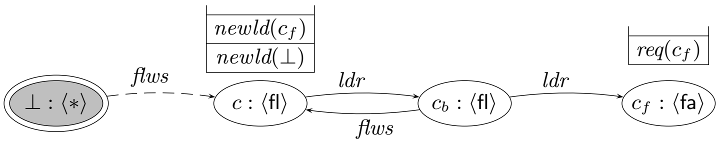
Fig. 1. Car Platooning: Two platoons and one free agent, from concrete to abstract.

Fig. 1(a)). Conceptionally, each car has a notion of *state*, that is, whether it is currently a *follower* in a platoon (like $c_{1,1}$), a *leader* of a platoon (like $c_{1,2}$), or whether it is driving on its own as a *free agent* (like $c_{3,3}$). In addition, there are *links* between cars, indicated by the arrows in Fig. 1(a). Each follower knows its leader, e.g. to negotiate leaving the platoon, each leader knows its followers, e.g. to announce a braking manoeuvre, and each car may know a lane-neighbour. Interlinked cars communicate by message passing. Note that we cannot assume a finite upper bound on the number of cars, as cars freely enter and leave the system "highway".

Example Requirement. There are three elementary actions for car platooning: *merge* and *split*, to build up and separate platoons, and *change lane*. In the following we'll focus on the merge action. If a platoon led by car c_b merges with a platoon in front led by car c_f, then during the merge c_b hands over its followers to c_f. For example, car $c_{3,2}$ merging with $c_{3,3}$ in Fig. 1(a) were an instance of this situation with $c_b := c_{3,2}$ and $c_f := c_{3,3}$. Given a formal DCS protocol model of merge (cf. Sect. 2), a natural requirement to check would concern this handover. Namely, for each follower c of c_b, whenever c_b sends a message '*newld*' (new leader) to c carrying the identity of the new leader c_f as a parameter, then c will finally change its leader link to c_f. Until that point in time, c_b remains c's leader. In the logic Mett (cf. Table 1 on page 7), this property is written as

$$\forall\, c_b, c, c_f \,.\, \underbrace{\mathsf{G}\ (\mathit{snd}[\mathit{newld}](c_b, c, c_f) \rightarrow (\mathit{conn}[\mathit{ldr}](c, c_b)\ \mathsf{U}\ \mathit{conn}[\mathit{ldr}](c, c_f)))}_{=:\mu_0} . \quad (1)$$

Analysis Problem. Faithfully modelling car platooning requires unbounded creation and destruction of processes as we cannot assume finite bounds on the



$newld(c_f)$
$newld(\bot)$
$req(c_f)$
flws
ldr
ldr
flws
$\bot : \langle * \rangle$
$c : \langle \mathsf{fl} \rangle$
$c_b : \langle \mathsf{fl} \rangle$
$c_f : \langle \mathsf{fa} \rangle$

Fig. 2. (Spurious) counter-example: Back leader c_b detected free agent c_f ahead and merged. Summary process $\bot$ sent a spurious '*newld*($\bot$)' before c_b correctly sent '*newld*(c_f)' to follower c. Consumption of the former by c leads to a violation of (1).

number of cars. DCS are consequently infinite-state systems. In order to employ automated verification techniques for temporal properties, we use a particular abstraction to finite-state transition systems following the *spotlight principle* [18]. The principle is to keep a finite number of processes completely precise and abstract from all others, the *rest*. In the particular case of Data Type Reduction (DTR, cf. Section 3), the number heuristically depends on the considered property and information about the state of the rest is completely dismissed, in particular links from the rest into the spotlight. Links from the spotlight are preserved but may point to the rest. Figures 1(c) and 1(d) illustrate the abstract state that represents Fig. 1(b), if the spotlight is on cars $c_{2,1}, c_{2,2}, c_{2,3}$. Having only the *local* information of the precise processes has the positive effect that the transition relation on abstract states is easily computable from a DCS description [19]. A negative effect is that the aggressive abstraction gives rise to a large amount of spurious behaviour, possibly comprising spurious counter-examples for a given property. As discussed in more detail in Section 3, many critical spurious runs, i.e. runs leading to a bad state in the abstract system, which is unreachable in the concrete one, follow a pattern we call spurious interference. In instances of this pattern, one observes messages from the abstracted *rest* to the concrete part, which are not possible in the concrete system.

For example, consider property (1) with three precise processes and the *rest* as in Fig. 1(d). If the *rest* sends a spurious '*newld*' message with an identity different from c_b and c_f to c, the property is violated (cf. Fig. 2). This situation can manually be identified as spurious by considering the DCS protocol description. By the DCS protocol, there is at most one car which may send '*newld*', namely the back leader. Since this is c_b, there cannot be a car in the *rest* sending the message. With this insight, it is desirable to refine the DTR abstraction by explicitly excluding such *communication topologies*, i.e. a global state of a DCS comprising processes connected by links, like the one shown in Fig. 2. Automating this kind of refinement poses two problems: (i) to automatically obtain information on the set of possible topologies; and (ii) to soundly exclude topologies shown impossible by (i).

Approach. Along this line of reasoning, the contributions of this paper are solutions to problems (i) and (ii). We tackle problem (i) by employing a new static analysis of graph grammars [4], called *Topology Analysis (TA)* below. Using our new encoding of DCS in graph grammars as presented in Section 4, we can thus compute abstract graphs, *topology invariants*, describing an over-approximation of all topologies possibly occurring in runs of a DCS.

Regarding problem (ii), Section 5 provides a logical characterisation of whether an abstract DTR topology like the one shown in Fig. 1(d) is definitely impossible according to the topology invariants computed by TA. These logical formulae can then be used as negative assumptions when model-checking the abstract transition system obtained by DTR. Here the challenge is that both the topologies in the abstract, finite-state transition system obtained by DTR and the topology invariants computed by TA are elements of *different abstract domains*. It is neither obvious how these domains relate formally nor whether an element of one domain represents more concrete topologies than an element of the other domain, since both typically represent infinitely many concrete topologies.

On top of first experimental results in Section 5 proving our approach to be effective, we briefly discuss in Section 6 whether the refinement proposed here could be further refined by, e.g., counter-example guidance.

Related Work. We specify DCS using DCS protocols (cf. Section 2). They were originally inspired by Communicating Finite State Machines (CFSM) of [5] but extend those by dynamic process creation and destruction and flexible communication topologies. Other than DCS protocols, DCS may be modelled using existing techniques such as the π-calculus [14] or variants of I/O automata (see [12] for an overview). The π-calculus is less adequate for our purpose, because crucial high-level features of DCS, like process states, message queues, or explicit graph-like communication topologies require cumbersome and low-level *encodings* into elementary π-actions. In that case, higher-level properties of a DCS are no longer accessible for specially tailored analyses or optimisations. Similar arguments hold for versions of I/O automata dealing with dynamic process creation, although they are admittedly much closer to DCS protocols. However, the better part of research on I/O automata is devoted to features like time or probability, while DCS protocols emphasise the dynamics of DCS.

DTR (cf. Section 3) is an instance of the *spotlight principle* [18] and as such related to different abstractions mapping infinite-state transition systems to finite-state ones comprising, e.g., [11] and [6]. DTR is unique in the coarseness of the abstraction, whose negative effects we remedy in this work.

Technically, Topology Analysis (TA, cf. Section 4) is an abstract interpretation of graph grammars, which are well suited to explicitly describe evolving DCS communication topologies. There is only one other abstract interpretation based approach to graph grammar verification [16]. Other approaches to graph grammar analysis use Petri-net based techniques [1]. However, these approaches are mostly concerned with the verification of pointer programs, where updates to graphs occur in a more restricted manner. If we chose π for modelling DCS, which we do not as justified above, [17] would offer a static analysis to determine communication topologies.

The principal idea of Section 5, to refine an abstraction with separately obtained invariants, is not new, e.g. [10], but the combination of TA and DTR is. Moreover, we integrate, in a technically sound way, two verification techniques often regarded as orthogonal: model checking and static program analysis. The synergy effect results in a novel automated verification technique for DCS.

It is noteworthy that hardly any of the specification and verification issues addressed in this work are addressed in the original PATH design [9], which deals with a static number of cars only. In particular, there are neither process creation, destruction, nor evolving communication topologies. In fact, in preliminary work [3], we manually discovered (and remedied) severe flaws in the original specification. With our proposed technique we are able to do so automatically.

2 Dynamic Communication Systems

Dynamic Communication Systems (DCS) can be viewed as a strict generalisation of classical parameterised systems because every process executes the same, finite control part, the *DCS protocol.* But in contrast to classical parameterised systems, where a fixed number of K processes run in parallel and communicate via global shared memory, processes are dynamically created and destroyed in DCS without an upper bound on the number of processes. Furthermore, DCS processes only have local memory, communicate asynchronously by passing messages, which may carry process identities, and in general aren't fully connected, but every process knows only some other processes via links.

In [3], we have introduced *DCS protocols* as an adequate modelling language for DCS, such as the car platooning example.

Syntax. A *DCS protocol* is a seven-tuple $\mathcal{P} = (Q, A, \Omega, \chi, \Sigma, \mathcal{E}_{\text{msg}}, \mathit{succ})$ with

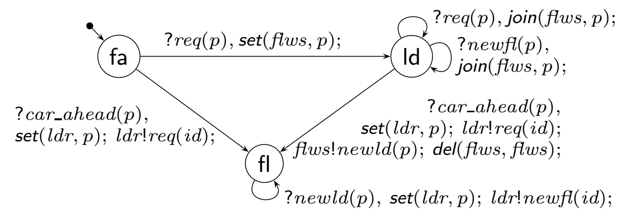
- a finite set Q of *states* a process may assume,
- *initial states* $A \subseteq Q$ assumed by newly appeared processes and *fragile states* $\Omega \subseteq Q$, in which processes may disappear,
- a finite set χ of *channels*, each providing potential links to other processes,
- a finite set Σ of *messages* and *environment messages* $\mathcal{E}_{\text{msg}} \subseteq \Sigma$, that is, messages that may non-deterministically be sent by the environment, and
- a *successor relation* '*succ*', determining each processes' behaviour.

The successor relation *succ* comprises four different kinds of labelled transitions between two states from Q, namely send, receive, modify, and conditional transitions. The corresponding (possibly empty) four sets partitioning *succ* are denoted by *Snd*, *Rec*, *Mod*, and *Cnd*. The notation for transitions is as follows.

- $(q, c, m, c', q') \in \mathit{Snd} \subseteq Q \times \chi \times \Sigma \times \chi \mathbin{\dot\cup} \{\mathit{id}\} \times Q$: send over channel c the message m carrying one of the identities stored in channel c' or the own identity if $c' = \mathit{id}$, and change to state q';
- $(q, m, c, \mathit{op}, q') \in \mathit{Rec} \subseteq Q \times \Sigma \times \chi \times \{\mathsf{set}, \mathsf{join}\} \times Q$: consume a message m, store the attached identity by operation op to channel c, change to state q';
- $(q, c_1, \mathit{op}, c_2, q') \in \mathit{Mod} \subseteq Q \times \chi \times \{\mathsf{add}, \mathsf{del}, \mathsf{pick}\} \times \chi \times Q$: combine channels c_1 and c_2 by operation op, store the result in channel c_1, change to state q';
- $(q, \mathit{em}, c, q') \in \mathit{Cnd} \subseteq Q \times \mathbb{B} \times \chi \times Q$: change to state q' if the emptiness constraint em on channel c is satisfied (see below).

Figure 3 shows a high-level model of the merge procedure which is supposed to be executed by each car. At any time, each car is either driving on its own as a *free agent* (fa) or participates in a platoon as *leader* (ld) or *follower* (fl).

$Q = \{\mathsf{fa}, \mathsf{ld}, \mathsf{fl}\}$
$A = \{\mathsf{fa}\}$, $\Omega = \emptyset$
$\chi = \{\mathit{flws}, \mathit{ldr}\}$
$\Sigma = \{\mathit{car\_ahead}, \mathit{req},$
$\quad \mathit{newld}, \mathit{newfl}\}$
$\mathcal{E}_{\text{msg}} = \{\mathit{car\_ahead}\}$



fa
ld
fl
?req(p), set(flws, p);
?req(p), join(flws, p);
?newfl(p), join(flws, p);
?car_ahead(p), set(ldr, p); ldr!req(id);
?car_ahead(p), set(ldr, p); ldr!req(id); flws!newld(p); del(flws, flws);
?newld(p), set(ldr, p); ldr!newfl(id);

Fig. 3. High-level implementation of "**platoon merge**".

Each follower knows its leader by the channel *ldr* and each leader knows its followers by the channel *flws*. For conciseness, the figure makes use of *complex transitions* [15] where one receive and a couple of send actions actually belonging to single transitions are being executed atomically.

The environment message '*car_ahead*' models that some sensors of a free agent discover cars in front and notify the free agent of the identity of such approached cars. The free agent then requests the merge by sending a message '*req*' which carries its identity and becomes a follower by taking the transition to fl. If a free agent receives a merge request, it stores the requester's identity in channel *flws* and changes state to ld. A leader may accept more followers and add them to *flws*, or merge with another free agent or platoon in front. In the latter case, it announces the new leader by a '*newld*' message to its followers and dismisses them, the followers then register with the new leader by a '*newfl*' message.

Semantics. Given a DCS protocol $\mathcal{P}$ and a countably infinite set *Id* of identities, a *(local) configuration* of a process is a triple (q, C, M) where $q \in Q$ is the local state, $C : \chi \to 2^{Id}$ is a function mapping channels to a set of process identities, and $M = (\Sigma \times Id)^*$ is the message queue. We use $\mathcal{S}(\mathcal{P})$ to denote the set of all local configurations of protocol $\mathcal{P}$. A local configuration (q, C, M) is called *initial* if $q \in A$, C yields the empty set for all channels, and M is the empty word.

A *topology* (or *global configuration*) of $\mathcal{P}$ is a partial function $\mathcal{N} : Id \rightharpoonup \mathcal{S}(\mathcal{P})$ mapping identities to local configurations. In the following we write $\iota \in \mathcal{N}$ to denote that $\mathcal{N}$ is defined for ι. A topology $\mathcal{N}$ *evolves* into $\mathcal{N}'$, written as $\mathcal{N} \rightsquigarrow \mathcal{N}'$, if one of the following conditions is satisfied, where all processes not affected by the evolution are required to remain the same. Note that the first and the last two actions are *environment actions* which are always enabled.

Appearance: A new process is created, starting in an initial state, not connected to anyone else, and with an empty queue, i.e. $\mathcal{N}' = \mathcal{N}[\iota \mapsto (q, C, M)]$ where $dom(\mathcal{N}') = dom(\mathcal{N}) \setminus \{\iota\}$ and configuration (q, C, M) is initial.

SendMessage: A process $\iota \in \mathcal{N}$ takes a send transition (q, c, m, c', q') and thus appends message m carrying an identity as parameter to the message queues of the processes denoted by its channel c, i.e. if $\mathcal{N}(\iota) = (q, C, M)$ we have that $\mathcal{N}'(\iota) = (q', C, M)$ and $\mathcal{N}'(\iota_0) = (q_0, C_0, M_0.(m, \iota'))$ for all $\iota_0 \in C(c)$ with $\mathcal{N}(\iota_0) = (q_0, C_0, M_0)$. If $c' = id$, the process sends its own identity, i.e. $\iota' = \iota$, otherwise it sends an element from channel c', i.e. $\iota' \in C(c')$.

$p_1 = p_2$	equality
$instate[q](p)$	in state q
$conn[c](p_1, p_2)$	linked via channel c
$\odot\, p$	just created

$snd[m](p_1, p_2, p)$	message m just sent
$pend[m](p_1, p_2, p)$	message m pending
$rcv[m](p_1, p_2, p)$	message m received
$\otimes\, p$	about to die

Table 1. Mett predicates (p, p_1, p_2 are quantified variables).

ReceiveMessage: A process $\iota \in \mathcal{N}$ takes a receive transition (q, m, c, op, q') and thus stores the identity carried by the message to channel c, i.e. if $\mathcal{N}(\iota) = (q, C, (m, \iota_0).M)$ we have $\mathcal{N}'(\iota) = (q', C', M)$ where $C'(c) = \{\iota_0\}$ if $op = \mathsf{set}$, and $C'(c) = C(c) \cup \{\iota_0\}$ if $op = \mathsf{join}$, and $C'(c') = C(c')$ for all $c' \neq c$.

ModifyChannel: A process $\iota \in \mathcal{N}$ takes a modify transition (q, c_1, op, c_2, q'), i.e. if $\mathcal{N}(\iota) = (q, C, M)$ we have $\mathcal{N}'(\iota) = (q', C', M)$ where $C'(c_1) = C(c_1) \cup C(c_2)$ if $op = \mathsf{add}$, and $C'(c_1) = C(c_1) \setminus C(c_2)$ if $op = \mathsf{del}$, and $C'(c_1) = \{\iota_0\}$ for some $\iota_0 \in C(c_2)$ if $op = \mathsf{pick}$, and $C'(c) = C(c)$ for all $c \neq c_1$.

Conditional: A process $\iota \in \mathcal{N}$ takes a conditional transition (q, em, c, q'), i.e. if $\mathcal{N}(\iota) = (q, C, M)$ such that $em \leftrightarrow (C(c) = \emptyset)$, then $\mathcal{N}'(\iota) = (q', C, M)$.

SendEnvMessage: A process $\iota \in \mathcal{N}$ obtains a message from $\mathcal{E}_{\mathrm{msg}}$ with an arbitrary identity from $\mathcal{N}$ attached similarly to the sending case above.

Disappearance: A process $\iota \in \mathcal{N}$ in a fragile state is destroyed, i.e. if $\mathcal{N}(\iota) = (q, C, M)$ and $q \in \Omega$ then $\mathcal{N}' = \mathcal{N}\mid_{dom(\mathcal{N}) \setminus \{\iota\}}$. Process ι then disappears from all channels, i.e. for all processes $\iota' \in \mathcal{N}'$ with $\mathcal{N}(\iota') = (q, C, M)$ we have $\mathcal{N}'(\iota') = (q, C', M)$ with $C'(c) = C(c) \setminus \{\iota\}$ for all channels c.

The *semantics* of $\mathcal{P}$ is the transition system with the (in general) infinite set of (unbounded) topologies of $\mathcal{P}$ as states, the empty topology as initial state, and transitions given by the evolution relation $\rightsquigarrow$ as defined above. That is, Fig. 3 models that cars freely enter the highway, without an upper bound on the number of cars. Triggered by the environment message '*car_ahead*', free agents or platoons then merge into platoons forming topologies of interlinked cars.

Requirements Specification Language. In [3], DCS is complemented by the requirements specification language Mett. A first example of a Mett requirement has already been given in Section 1 in formula (1). Mett is basically a first-order extension of LTL providing quantification over anonymous objects and the predicates given by Table 1, which refer to processes' local state, topology, and communication. Given a Mett formula μ and a DCS protocol $\mathcal{P}$, the satisfaction relation $\mathcal{P} \models \mu$ is inductively defined on the transition system induced by $\mathcal{P}$. The definition is standard and omitted for space reasons.

3 Data Type Reduction

By the definitions in Section 2, we have to deal with three dimensions of complexity when verifying Mett properties for DCS: finite control within each process, unbounded, evolving topologies of processes, and queue based communication. Here, we focus on the first two layers and restrict message queues to length 1.

Data Type Reduction (DTR) is an abstraction technique, which has originally been introduced for the verification of properties of parameterised systems [13]. In [7], it has been demonstrated that this abstraction applies as well to systems with unbounded dynamic creation and destruction of processes, thus in particular to DCS. Beyond [7, 13], for space reasons, we only recall as much of DTR as is necessary to understand the intrinsic problem of spurious counterexamples to be cured with topology invariants (cf. Section 4).

DTR actually applies to quantifier-free properties with free variables, like μ_0 of (1) on page 2. The universally quantified case, i.e. the whole formula (1), follows by symmetry from finitely many cases [13]. DTR employs the spotlight principle [18], that is, it maps each concrete topology $\mathcal{N}$ to an *abstract topology* where a fixed number of processes is kept completely precise and information about the rest is lost (cf. Fig. 1). An abstract topology $\mathcal{N}_N^\sharp$ maps the limited set of identities $Id_N^\sharp = \{u_1, \ldots, u_N, \bot\}$, where N is the number of individuals kept precise, to local configurations from $\mathcal{S}(\mathcal{P})$. Thereby, the set of identities is reduced to a finite set and consequently there are only finitely many abstract topologies. The abstraction of a concrete topology like Fig. 1(d) is obtained from Fig. 1(b) by replacing all identities, in links and messages, belonging to the rest by the special identity $\bot$. The local configuration of $\bot$ is the upper bound of all *possible* local configurations, not only of all the ones present in the original topology; this is graphically illustrated for links by dashed arrows in Fig. 1(d). Existential abstraction of the original system's transition relation yields a finite-state transition system with abstract topologies as states (cf. [18]). Identity $\bot$ is special as it compares inconclusive to itself and unequal to others, i.e. when the leaders of two cars are compared and are $\bot$, then both possible outcomes are explored. With $\mathcal{P}_N^\sharp, \theta \models \mu_0$ denoting that the METT formula μ_0 is valid under assignment θ in the abstract transition system induced by DTR for $\mathcal{P}$ and $N \in \mathbb{N}$, which can heuristically be chosen depending on the formula, we have

Lemma 1 (Soundness of DTR [7]). *Given a DCS protocol $\mathcal{P}$ and a quantifier-free* METT *formula μ_0 over variables $p_1, \ldots, p_n$, DTR is sound for any $N \in \mathbb{N}$ and assignment $\theta : \{p_1, \ldots, p_n\} \to Id_N^\sharp \setminus \{\bot\}$, i.e. $\mathcal{P}_N^\sharp, \theta \models \mu_0 \Rightarrow \mathcal{P}, \theta \models \mu_0$.* $\diamondsuit$

The abstract transition relation is easily obtained by a syntactical transformation of the DCS protocol because only information *local* to the finitely many processes in the spotlight has to be represented [19]. This property is easily lost when trying to explicitly add precision to the abstract topologies.

However, the abstraction is rather coarse allowing for many spurious interference initiated from the shadows as described in Section 1. There, spuriousness results from the $\bot$ process sending a '*newld*' message with an unexpected identity attached. We explained that this could only happen in topologies that are in fact impossible for the given DCS protocol.

4 Topology Analysis

As outlined in Section 1, the idea presented in this paper is to add precision to the rather coarse abstract transition system obtained by DTR (cf. Section 3) by forcing it to adhere to certain, independently established, invariants. In other words, to let the abstract system avoid definitively illegal topologies.

A particular approach to obtain information about legal topologies is Topology Analysis (TA) [4]. Its subject are directed node- and edge-labelled graphs and graph grammars, that is, sets of graph transformation rules. The static analysis of a graph grammar yields a rather precise finite over-approximation, called *topology invariant*, of all graphs possibly generated by the graph grammar. Formally, topology invariants are sets of *abstract clusters*. An instance of an abstract cluster is any graph that can be abstracted to it by *partner abstraction*. Partner abstraction in turn is quotient graph building with respect to partner equivalence, which is motivated by preserving information about *who is talking when to whom*. Intuitively, two processes are partner equivalent if they are in the same state and if they have links to the same kind of processes — regardless of the number of such communication partners. In the context of dynamic *communication* systems, such information is valuable, because it is really this information that determines possible successor topologies of a given topology.

The following paragraph formally rephrases the absolute necessities of the technique of [4]. The subsequent paragraphs contribute an encoding of DCS into graph grammars, thereby making TA amenable to DCS verification.

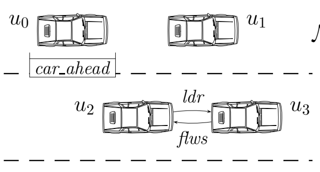
Topology Analysis. A *graph* is a five-tuple $G = (V, E, s, t, \ell)$ featuring a set of nodes, a set of edges, a source-, a target-, and a labelling function. Source and target functions map edges to their respective source and target nodes, while ℓ maps both, nodes and edges, to labels. A *graph grammar* $\mathfrak{G}$ is a finite set of graph transformation rules. A graph transformation rule consists of two graphs, a *left graph* L and a *right graph* R, and a relation between them indicating which nodes and edges in L and R correspond to each other. In the rule shown in Fig. 5, this correspondence is given implicitly by position. A rule can be *applied* to a graph G, if L is a subgraph of G. The result of the application is the replacement of L's occurrence in G with R. For more details we refer to [4].

Two nodes $u_1, u_2 \in V$ are partner equivalent if $\ell(u_1) = \ell(u_2)$ and if for all edge labels a, $o_G(a, u_1) = o_G(a, u_2)$ and $i_G(a, u_1) = i_G(a, u_2)$. These sets denote the labels of the nodes adjacent to u_1 and u_2, that is $o_G(a, u_1) := \{\ell(v) \mid \exists e \in E : s(e) = u_1, t(e) = v, \ell(e) = a\}$ and analogously for incoming edges.

Given a graph G, the *abstract cluster* $\alpha_{TA}(G)$ is an abstraction of G. It is computed in two steps: First, for each connected component C of G compute the quotient graph with respect to partner equivalence. Doing so, mark equivalence classes consisting of more than one node as a *summary node*. As a second step, summarise isomorphic quotient graphs, that is, keep only one of them.

The set of abstract clusters obtained by the abstract interpretation based on partner abstraction for a graph grammar $\mathfrak{G}$ and the empty graph as initial graph is called *topology invariant* of $\mathfrak{G}$ and denoted by $\mathscr{G}_{\mathfrak{G}}$.

Lemma 2 (Soundness of TA [4]). *Let $\mathfrak{G}$ be a graph grammar and graph G obtained from the empty graph by applying $\mathfrak{G}$. Then $\alpha_{TA}(G) \subseteq \mathscr{G}_{\mathfrak{G}}$.* $\diamondsuit$



$$\begin{aligned}\mathcal{N} = \{&\iota_0 \mapsto (\mathsf{fa}, [\,], (car\_ahead, u_1)),\\ &\iota_1 \mapsto (\mathsf{fa}, [\,], \epsilon),\\ &\iota_2 \mapsto (\mathsf{fl}, [ldr \mapsto \{u_3\}], \epsilon),\\ &\iota_3 \mapsto (\mathsf{ld}, [flws \mapsto \{u_2\}], \epsilon)\}\end{aligned}$$

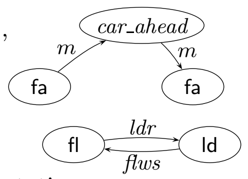


Fig. 4. DCS topology $\mathcal{N}$ and its graph representation.

DCS Topologies as Graphs. Let $\mathcal{P} = (Q, A, \Omega, \chi, \Sigma, \mathcal{E}_{\text{msg}}, succ)$ be a DCS protocol. A topology $\mathcal{N} : Id \rightharpoonup \mathcal{S}(\mathcal{P})$ of $\mathcal{P}$ is encoded as a directed, node- and edge-labelled graph $T(\mathcal{N})$ as follows. For each process in $dom(\mathcal{N})$ and each message in one of the processes' message queues, there is a node. Nodes representing processes are labelled with their local state $q \in Q$, nodes representing messages are labelled with the message name $e \in \Sigma$. For each process u and each channel $c \in \chi$, there are edges labelled with c to each element of $C(c)$.

For each message, there is an edge labelled m from the destination to the message node and from the message to its parameter (cf. Fig. 4). Note that this representation of queues is only feasible for our restriction to finite queue lengths. Unbounded message queues would have to be properly encoded as lists.

Actions as Graph Transformation Rules. Each element of *succ* is translated into a set of graph transformation rules. An example of a graph transformation rule resulting from the "send identity" action $(\mathsf{ld}, req, ldr, id, \mathsf{fl}) \in Snd$ is shown in Fig. 5. The left graph shows a situation where a process u_1 in state ld is connected to process u_2 via channel ldr. The result of u_1 sending its own identity attached to message req to process u_2 is exactly the right graph. Process u_1 has changed its state to fl, u_2 has an m-labelled edge to message node u_m, and message node u_m is connected to process u_1. Similarly, environment interaction like process creation or environment messages translates into graph transformation rules. Note that this translation of a DCS protocol into graph transformation rules can be conducted fully automatically.

Topology Invariants. Applying Topology Analysis to the encoding of the platoon merge DCS yields a topology invariant which comprises in particular the four abstract clusters shown in Fig. 6. Intuitively, an abstract cluster denotes that in each topology there may be arbitrary many, that is zero or more, instances of it. For example, $C_1^\sharp$ denotes the possibility of arbitrary many free agents in each topology of the platoon merge DCS. In Fig. 6, summary nodes are drawn

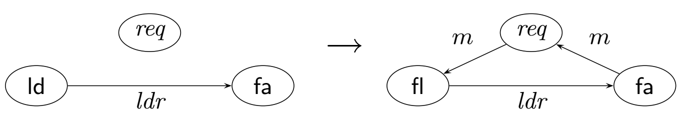


Fig. 5. Graph transformation rule: a platoon approaches a free agent, the platoon leader is in state ld and sends a req message to the free agent in front with its identity as parameter. Afterwards the former leader is in state fl.

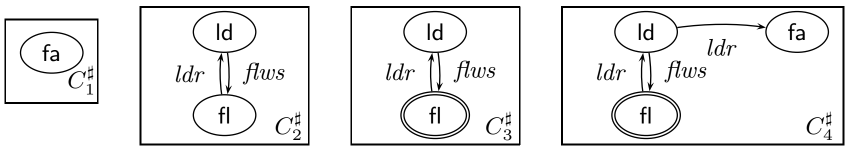


Fig. 6. Abstract clusters. Doubly outlined nodes are summary nodes.

with a double-outline. So abstract cluster $C_2^\sharp$ represents an arbitrary number of platoons of size two, whereas $C_3^\sharp$ represents an arbitrary number of platoons of size greater than two. Altogether, a topology invariant denotes any combination of possible instances of the abstract clusters belonging to it.

In terms of a DCS protocol $\mathcal{P} = (Q, A, \Omega, \chi, \Sigma, \mathcal{E}_{\text{msg}}, succ)$, a topology invariant is a set $\mathscr{G}_\mathcal{P} = \{C_1^\sharp, \ldots, C_n^\sharp\}$ of abstract clusters. The labelling ℓ_i of $C_i^\sharp$ labels nodes with pairs (st, sm) where st is either a DCS protocol state from Q or a message from Σ. The boolean flag sm indicates whether the node is a summary node or not. Each edge in E_i is labelled by ℓ_i with a channel from χ. Given this encoding we can apply Lemma 2 and obtain:

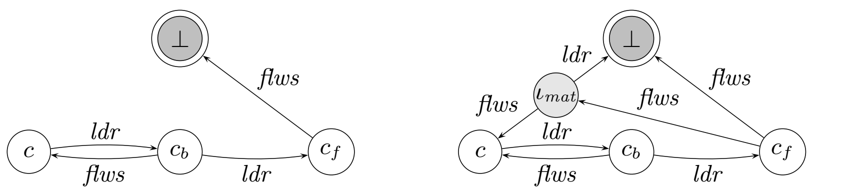
Corollary 1. *Given DCS protocol $\mathcal{P}$ and topology $\mathcal{N}$ of $\mathcal{P}$, $\alpha(T(\mathcal{N})) \subseteq \mathscr{G}_\mathcal{P}$.* $\diamondsuit$

It proves to be a difficult task to relate the knowledge about valid clusters with the abstract topologies obtained by DTR. We explain our approach to this problem being one of the major contributions of this work in the next section.

5 Putting it Together: Respecting Topology Invariants

Recall from Section 3 that DTR yields a finite-state transition system operating on abstract topologies $\mathcal{N}^\sharp$. They have the form shown in Fig. 1(d), that is, finitely many concrete processes and a summary node representing the rest.

Materialisation. We have briefly discussed in Sections 1 and 3 that this coarse abstraction gives rise to spurious behaviour which can in many cases (manually) be identified as spurious by examining the abstract topologies. This examination is based on the observation of messages that are sent from the summarised rest to concrete processes. For example, consider the abstract topology shown in Fig. 7(a), which is the same as the one in Fig. 2. If in this abstract topology $\mathcal{N}^\sharp$, the summary process $\bot$ sends a message '$newld(\bot)$' to c, we can conclude that the concretisation of $\mathcal{N}^\sharp$ comprises a topology where there is a process capable of sending such a message. Inspecting the DCS protocol (cf. Fig. 3), we see that there must be at least one process ι_{mat} in the rest, which is in state ld, which necessarily has a follower link to c to know the destination, and which has a leader link either to $\bot$ or to itself to know the sent identity. Concerning the link from c_f to $\bot$, we cannot definitely conclude whether it has a link to only one or both of the grayish nodes, that is, there are three cases. Adding all this information to Fig. 7(a) yields Fig. 7(b), which shows one of the six possible materialisations. Figuratively speaking, the view on an abstract



(a) Abstract topology. (b) Light gray individual materialised.

Fig. 7. Materialisation. When a message is sent from the summary process (the dark gray individual), we can conclude on a part of the topology. In the example, we can conclude that there has to be a process having c as follower and c_f as leader.

topology changes from one with only a single "dark gray" summary process to one with an additional "light gray" process that has been materialised from the summary process. Note that the materialised process is still "gray" in the sense that we typically don't know everything about its configuration but only about the parts involved in the observed action. For example, we cannot conclude whether ι_{mat} has more followers than c.

More formally, if we're in abstract state $\mathcal{N}^\sharp$, representing topology $\mathcal{N}$, and action $ac = (q, c, m, c', q') \in \mathit{Snd}$ is to be executed by the process summary $\perp$ we can conclude that (i) there is a process ι_{mat} summarised by $\perp$, which has local state q, (ii) this process ι_{mat} has ι in channel c, and (iii) it has ι' in channel c'.

Here, the topology invariants automatically established by Topology Analysis (cf. Section 4) come into play as follows. For each abstract topology $\mathcal{N}^\sharp$ in a run of the abstract transition system and each send action $ac \in \mathit{Snd}$ we can derive a (finite) set $\mathit{mat}(\mathcal{N}^\sharp, ac)$ of materialisations following the reasoning above. The run is spurious, if it employs at least one abstract topology and action such that all materialisations are *definitely illegal*. We call a materialisation as the one in Fig. 7(b) definitely illegal if it is *contradictory* to topology invariant $\mathscr{G}_\mathcal{P}$. For this, note that a materialisation $\mathcal{N}^\sharp_{mat}$ again represents a set of concrete topologies which we denote by $\gamma_{DTR}(\mathcal{N}^\sharp_{mat})$. Then materialisation $\mathcal{N}^\sharp_{mat}$ is contradictory to $\mathscr{G}_\mathcal{P}$ iff $\alpha_{TA}(T(\gamma_{DTR}(\mathcal{N}^\sharp_{mat}))) \not\subseteq \mathscr{G}_\mathcal{P}$. Note that this definition doesn't directly yield a decision procedure because $\gamma_{DTR}(\mathcal{N}^\sharp_{mat})$ is in general an infinite set.

Logical Characterisation of Topology Invariants. In the following, we present a solution which isn't based on computing the infinite concretisations but employs a kind of unification between materialised abstract topologies and abstract clusters. This unification is expressed in terms of existentially quantifying predicate logic formulae to be evaluated over materialisations in the 3-valued Kleene interpretation of logic. Using 3-valued logic we treat the fact that materialised nodes are still "light gray", that is, we may not know all of their attributes. References to other predicates of the car, e.g. whether it is currently accelerating, would yield $1/2$ as the definite value cannot be concluded from the observed communication behaviour. Indefiniteness propagates naturally over logical connective, that is, $(1 \wedge 1/2)$ yields $1/2$, while $(0 \wedge 1/2)$ yields 0. A formula evaluating to 1 then indicates that a materialisation is definitely feasible, in case of $1/2$, it is possibly feasible, and in case of 0 it is definitely impossible.

After materialisation, we have finite sets of abstract topologies and abstract clusters $\mathscr{G}_{\mathcal{P}}$ obtained by Topology Analysis, that is, two kinds of graphs with summary nodes that may match in numerous ways. We only consider the white and light gray processes in the materialisations and individually check whether *their* situation is possible according to $\mathscr{G}_{\mathcal{P}}$. In other words, if the situation of a white or light gray process in a materialisation $\mathcal{N}^{\sharp}_{mat}$ doesn't occur in $\mathscr{G}_{\mathcal{P}}$, then none of the concretisations of $\mathcal{N}^{\sharp}_{mat}$ are feasible in the concrete system (by Corollary 1). Then if none of the materialisations from $mat(\mathcal{N}^{\sharp}, ac)$ are feasible, all system runs on which ac applied to $\mathcal{N}^{\sharp}$ is observed are spurious. We call an abstract topology materialisation $\mathcal{N}^{\sharp}_{mat}$ *possibly legal* wrt. $\mathscr{G}_{\mathcal{P}}$ iff for each white and light gray process $\iota \in \mathcal{N}^{\sharp}$ there is an abstract cluster $C^{\sharp} = (V, E, s, t, \ell) \in \mathscr{G}_{\mathcal{P}}$ and a *matching* node $v \in V$ such that (a) $\ell(v)$ is the state of ι, (b) for each c-edge from v to v' there exists a process ι' in state $\ell(v')$ and the channel c of ι comprises ι', and (c) each c'-edge from v' to v implies that the channel c' of ι' comprises ι.

In order to respect the topology invariant $\mathscr{G}_{\mathcal{P}}$ during model-checking we express the relation induced by (a)–(c) by a logical formula. A process $\iota \neq \bot$ in a materialised abstract topology $\mathcal{N}^{\sharp}_{mat}$ has $v \in V$ as *matching node* iff there is a bijection $\beta : \{v' \in V \mid (v, v') \in E\} \to dom(\mathcal{N}^{\sharp}_{mat})$ such that

$$\phi_v(\mathcal{N}^{\sharp}_{mat}, \iota) := q(\iota) = \ell(v) \wedge \bigwedge_{dom(\beta)} \Big(q(\beta(v')) = \ell(v') \wedge \\ \bigwedge_{\{e \in E \mid s(e)=v \wedge t(e)=v'\}} \ell(e)(\iota, \beta(v')) \wedge \bigwedge_{\{e \in E \mid s(e)=v' \wedge t(e)=v\}} \ell(e)(\beta(v'), \iota) \Big) \quad (2)$$

is not 0, i.e. either 1 or $1/2$ (see above), where $q(\iota)$ denotes the state of process ι and $c(\iota, \iota')$ yields true iff there is a link c from ι to ι' in $\mathcal{N}^{\sharp}_{mat}$. Lifting this characterisation to the level of the whole abstract cluster, the process ι is possibly legal according to $C^{\sharp}$ if one node of $C^{\sharp}$ matches, i.e. if

$$\phi_{C^{\sharp}}(\mathcal{N}^{\sharp}_{mat}, \iota) := \exists\, v \in V : \phi_v(\mathcal{N}^{\sharp}_{mat}, \iota) \quad (3)$$

is not 0. A process ι in $\mathcal{N}^{\sharp}_{mat}$ is possibly legal according to $\mathscr{G}_{\mathcal{P}}$ if it is possible legal according to one of the abstract clusters, i.e. if

$$\phi_{\mathscr{G}_{\mathcal{P}}}(\mathcal{N}^{\sharp}_{mat}, \iota) := \exists\, C^{\sharp} \in \mathscr{G}_{\mathcal{P}} : \phi_{C^{\sharp}}(\mathcal{N}^{\sharp}_{mat}, \iota) \quad (4)$$

is not 0. Finally, the whole materialised abstract topology $\mathcal{N}^{\sharp}_{mat}$ is possibly legal according to $\mathscr{G}_{\mathcal{P}}$ if all processes are possibly legal, i.e. if

$$\phi_{\mathscr{G}_{\mathcal{P}}}(\mathcal{N}^{\sharp}_{mat}) := \forall\, \iota \in dom(\mathcal{N}^{\sharp}_{mat}) \setminus \{\bot\} : \phi_{\mathscr{G}_{\mathcal{P}}}(\mathcal{N}^{\sharp}_{mat}, \iota). \quad (5)$$

is not 0. Note that the quantifications in (3) - (5) are over finite sets, thus expand to unquantified predicate logic expressions.

For example, let $\mathcal{N}^\sharp_{mat}$ denote the materialised abstract topology from Fig. 7(b). Than $\phi_{C^\sharp_1}(\mathcal{N}^\sharp_{mat}, \iota_{mat})$ evaluates to 0 for $C^\sharp_1$ from Fig. 6 because ι_{mat} is in state ld and no node in $C^\sharp_1$ has this label. Also $\phi_{C^\sharp_4}(\mathcal{N}^\sharp_{mat}, \iota_{mat})$ evaluates to 0 although there is a node labelled with ld in $C^\sharp_4$, but $C^\sharp_4$ requires that there is a leader link back to the ld-node from each follower, which is not the case for ι_{mat}. Actually, none of the abstract clusters in $\mathscr{G}_\mathcal{P}$ matches, thus Fig. 7(b) is definitely illegal, a run exhibiting it is spurious.

Assuming Adherence. The indication of spuriousness, namely executing an action *ac* in abstract topology $\mathcal{N}^\sharp$ whose effect would require the existence of an illegal topology in the concrete, is a non-temporal property depending only on $\mathcal{N}^\sharp$ and *ac*. As such it can easily be added as a boolean observer to an abstract transition system $\mathcal{P}^\sharp_N$. Then by $\mathcal{P}^\sharp_N, \theta, \mathscr{G}_\mathcal{P} \models \mu_0$, we denote that the abstract transition system obtained by DTR satisfies μ_0 on those system runs, where the observer doesn't indicate spuriousness and we have the following

Theorem 1 (Soundness of DTR+TA). *Let $\mathcal{P}$ be a DCS protocol and μ_0 a* METT *formula over variables $p_1, \dots, p_M$. Then given $N \in \mathbb{N}$ and an assignment $\theta = \{p_i \mapsto u_i\}$, DTR+TA is sound, i.e. $\mathcal{P}^\sharp_N, \theta, \mathscr{G}_\mathcal{P} \models \mu_0 \implies \mathcal{P}, \theta \models \mu_0$.* ♢

The proof is based on Corollary 1 by which the restriction to $\mathscr{G}_\mathcal{P}$ only removes transitions from $\mathcal{P}^\sharp_N$ which lead to topologies that aren't possible in the original transition system. Consequently, no original system behaviour is disregarded.

Note that the example discussed below indicates that we indeed obtain a *proper* refinement, i.e. there are properties that cannot be established in result of DTR but can in combination with topology invariants.

Experimental Results. For a proof-of-concept implementation, we had to strengthen the platoon-merge protocol in comparison to the one shown in Fig. 3. The handover of followers to the new leader on a merge, for instance, needs guarding acknowledge messages. This protocol provides for a topology invariant with 77 abstract clusters automatically computed by an implementation of the Topology Analysis [4]; a looser merge protocol easily yields 2000 clusters.[3]

We used the tool-set of [15] to translate the strengthened merge protocol into the input language of the VIS [8] model-checker. Most recently, we have implemented the translation of DCS protocols into graph transformation rules fully automating our tool chain. Without respecting the topology invariants, the model-checker unveils the counter-example discussed in the introduction in about 90 minutes. After encoding the topology invariant following the procedure from the previous section, we were able to prove the property in about 126 minutes. Note that Topology Analysis alone is not able to establish (1) because it comprises a liveness requirement and TA only addresses safety properties.

[3] For the full strengthened protocol and the actually employed clusters see App. A.

6 Conclusion

We promote a combination of an easily obtainable but rather coarse abstract transition system with external information obtained by static analysis. Our technique has a number of benefits. It can automatically prove properties that neither technique can prove in isolation demonstrating synergy. It formally integrates techniques, namely model checking and static program analysis, that are often considered orthogonal. Moreover, it is able to automatically discover flaws in sophisticated traffic control applications [9] that could only be found manually [3] before. Finally, the technique has been brought to full automation by integrating existing tools.

The technique introduced in Section 5 uses only a small fraction of the information carried by abstract clusters. Namely, for conciseness and to bound the size of the resulting formulae, we only considered *positive* links to distance-1 nodes, that is, the *direct* neighbourhood of nodes, as being relevant to exclude spurious behaviour. Further work comprises the evaluation of two aspects: looking further into the abstract cluster, i.e. to check whether processes up to some distance larger than 1 are legal according to the topology invariant, and considering information about the *absence* of links in abstract clusters.

Additionally, a kind of counter-example guided abstraction refinement could be established where each action of the summary process in the counter-example is checked for validity with respect to the topology invariant, and if one action turns out to be spurious, then in the next run only those abstract clusters are (soundly) considered that relate to the particular spurious interferences.

Regarding applications, we are currently implementing a number of examples from application domains such as mobile ad-hoc networks and service-oriented computing as DCS protocols. This is necessary to gather more experience on usability and scalability of our toolset. In order to facilitate the implementation of examples of realistic size, we plan to write a front-end automatically compiling more high level specifications, e.g. written in UML, into DCS protocols.

References

1. P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: An unfolding-based approach. In *Proc. CONCUR 2004*, volume 3170 of *LNCS*, 2004.
2. J. Bauer. *Analysis of Communication Topologies by Partner Abstraction.* PhD thesis, Universität des Saarlandes, 2006.
3. J. Bauer, I. Schaefer, T. Toben, and B. Westphal. Specification and Verification of Dynamic Communication Systems. In *Proc. ACSD 2006*. IEEE, June 2006.
4. J. Bauer and R. Wilhelm. Static Analysis of Dynamic Communication Systems by Partner Abstraction. In *Proc. SAS 2007*, 2007. To appear.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the Association for Computing Machinery*, 30(2):323–342, April 1983.
6. E. M. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proc. VMCAI '06*, volume 3855 of *LNCS*, pages 126–141, 2006.
7. W. Damm and B. Westphal. Live and let die: LSC-based verification of UML-models. *Science of Computer Programming*, 55(1–3):117–159, March 2005.

8. R.K.Brayton et.al. VIS: a system for verification and synthesis. In *Proc. CAV 1996*, number 1102 in LNCS, pages 428–432, 1996.
9. A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. The design of platoon maneuver protocols for IVHS. PATH Report UCB-ITS-PRR-91-6, U. California, April 1991.
10. H. Jain et al. Using statically computed invariants in the predicate abstraction and refinement loop. In *Proc. CAV 2006*, volume 4144 of *LNCS*, pages 137–151, 2006.
11. B. D. Lubachevsky. An approach to automating the verification of compact parallel coordination programs. *Acta Inf.*, 21:125–169, 1984.
12. N. A. Lynch. Input/output automata: Basic, timed, hybrid, probabilistic, dynamic, ... In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 187–188. Springer, 2003.
13. K. L. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37:279–309, 2000.
14. R. Milner. *The π Calculus.* Cambridge University Press, 1999.
15. J. Rakow. Verification of Dynamic Communication Systems. Master's thesis, Carl von Ossietzky Universität Oldenburg, April 2006.
16. A. Rensink and D. Distefano. Abstract graph transformation. *Electr. Notes Theor. Comput. Sci.*, 157(1):39–59, 2006.
17. A. Venet. Automatic determination of communication topologies in mobile systems. In Giorgio Levi, editor, *SAS*, volume 1503 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 1998.
18. B. Wachter and B. Westphal. The spotlight principle. In *Proc. VMCAI 2007*, volume 4349 of *LNCS*, pages 182–198, 2007.
19. B. Westphal. LSC verification for UML models with unbounded creation and destruction. In *Proc. SoftMC 2005*, volume 144(3) of *ENTCS*, pages 133–145, 2005.

Appendix A

Figure 8 shows the strengthened merge protocol in the DCS syntax as introduced in Section 2 which was used to obtain the experimental results. The main modification in contrast to the simple merge protocol from Fig. 3 is that state changes are guarded by exchanging *acknowledgement* messages, e.g. a free agent (state 'fa') sending a request does not immediately become a follower (state 'fl'), but moves to a state 'wait' to receive a proper confirmation for his request first. Additionally, the handing-over of followers when two platoons merge is separated into picking one follower into a separate channel '*aux*' and sending the '*newld*' message to this channel (transition 'busy' to 'pass'). After confirmation, the car is removed from the set of followers (transition 'pass' to 'rdy'). State 'rdy' then checks whether all followers have been handed over or not.

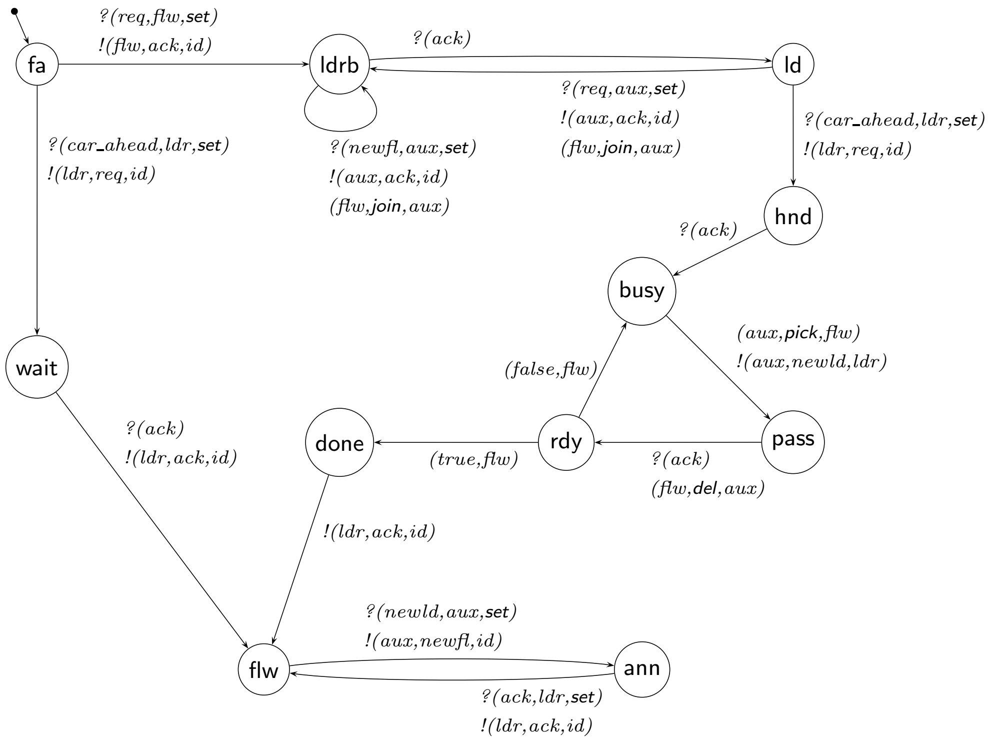


Fig. 8. The strengthened merge protocol.

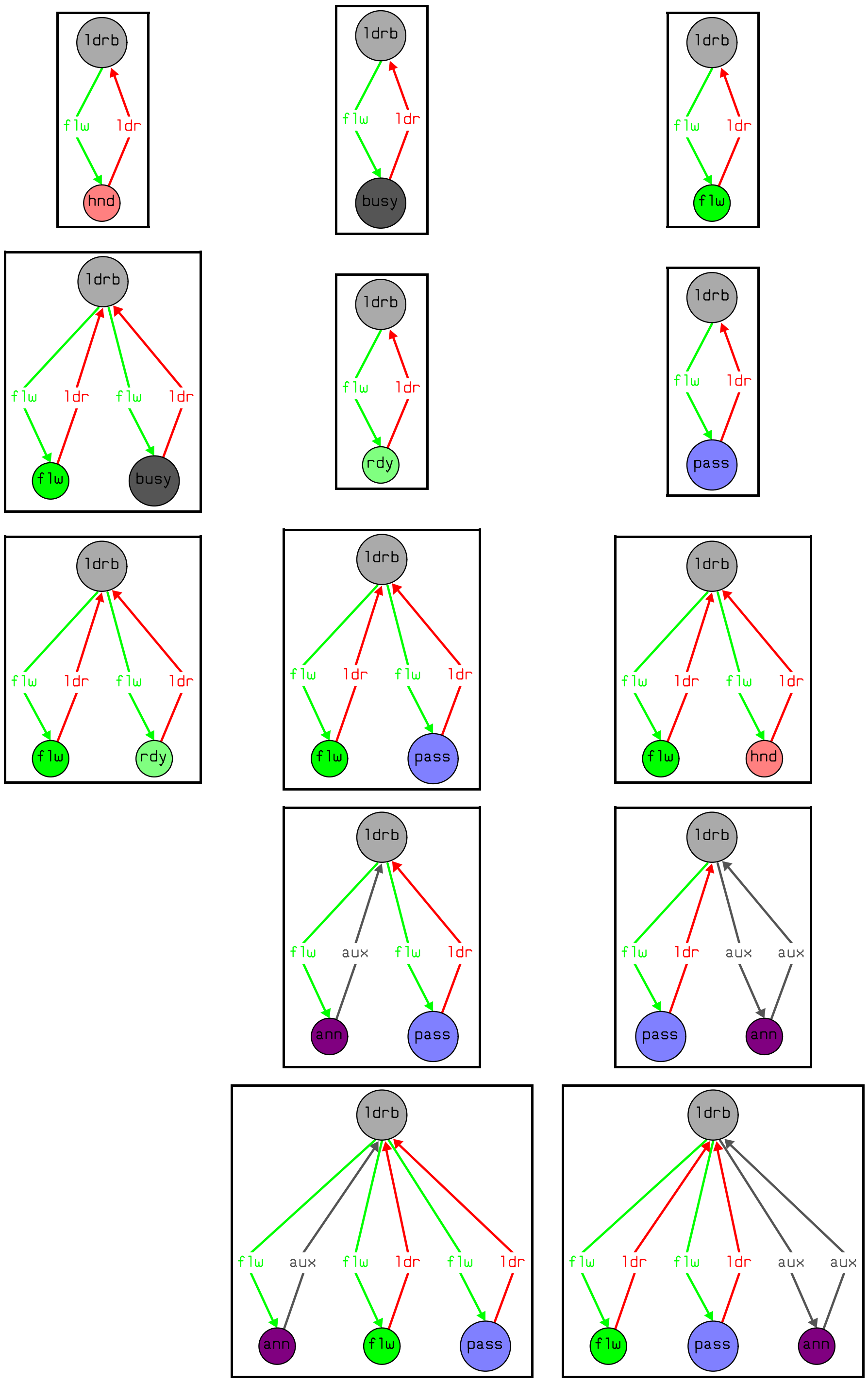


Fig. 9. Relevant part of the **topology invariant**, obtained by the tool-set of [2]. Only topologies where a process in state 'ldrb' is involved are shown. Also, all message nodes have been removed from the representation.