

# Non-Interference Properties for Data-Type Reduction of Communicating Systems<sup>\*</sup>

Tobe Toben

Carl von Ossietzky Universität Oldenburg, Germany.  
toben@informatik.uni-oldenburg.de

**Abstract.** An increasing interest in “Systems of Systems”, that is, Systems comprising a varying number of interconnected sub-systems, raises the need for automated verification techniques for dynamic process creation and a changing communication topology. In previous work, we developed a verification approach that is based on finitary abstraction via Data-Type Reduction. To be effective in practice, the abstraction has to be complemented by non-trivial assumptions about valid communication behaviour, so-called non-interference lemmata.

In this paper, we mechanise the generation and validation of these kind of non-interference properties by integrating ideas from communication observation and counter abstraction. We thereby provide a fully automatic procedure to substantially increase the precision of the abstraction.

We explain our approach in terms of a modelling language for dynamic communication systems, and use a running example of a car platooning system to demonstrate the effectiveness of our extensions.

## 1 Introduction

The current trend of *mobile computing* induces complex systems that comprise a varying number of interconnected sub-systems, for example in ad-hoc networking [6] where mobile devices detect other devices within a certain scanning range and autonomously maintain networks of arbitrary size. Similar principles are nowadays employed in traffic control systems, e.g. the European Train Control System [18] defines so-called Radio Block Controllers (RBC) along the track that are dynamically contacted by trains entering their communication range. From the viewpoint of an RBC, the responsibility for spontaneously appearing trains adds a new level of complexity when designing safety-critical systems.

Not surprisingly, automatic verification of systems with dynamic process creation and destruction is an active research problem. As the number of involved process within the system is a priori unbounded, finitary abstraction is applied to obtain a manageable representation of the infinite state system. As demonstrated in the literature [5, 20], in particular the technique of Data-Type Reduction (DTR) [12] is suitable for systems with dynamic process creation. Up to

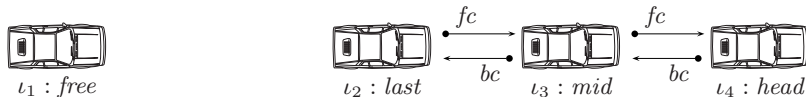
---

<sup>\*</sup> This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Centre “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS).

now, it is however an open problem how DTR can effectively be refined in order to reduce the large amount of spurious behaviour stemming from the abstraction.

In this paper, we devise an automatic procedure for excluding a typical source for spurious behaviour in abstractions of communicating systems, namely message interferences. We show that the observation of the communication history allows to identify invalid communication even if the actual status of a communication partner is blurred by the abstraction. We demonstrate our approach in terms of a modelling language for *Dynamic Communication Systems* (DCS) [2], a proper extension of *Communicating Finite State Machines* [3] by process creation and dynamic link topologies. The following example system will be used throughout the paper in order to demonstrate the ideas of our approach.

*Running Example “Car Platooning”.* Another real-world example of a DCS is the car platooning scenario as studied in the PATH project [9]. There, cars travelling on a highway are supposed to form *car platoons*, i.e. to establish series of interlinked cars driving with only little distance. The main manoeuvres are *merge*, i.e. a car joins an existing platoon, and *split*, i.e. a platoon is separated into unconnected cars again. Figure 1 depicts a snapshot of the car platooning system. Every car is aware of its current state, i.e. whether it is not involved in a platoon (*free*), the last car in a platoon (*last*), the first car in a platoon (*head*) or somewhere in the middle of a platoon (*mid*). The platoon itself is organised as a doubly-linked list of wireless communication channels, where a car knows its front car under the name *fc* and its back car under the name *bc*.



**Fig. 1.** Snapshot of the car platooning system where car  $\iota_1$  approaches a platoon. After detecting and merging with car  $\iota_2$ , a platoon of size four will be established.

*Analysis Approach.* Due to the unbounded number of participating processes, DCS are infinite state systems. In [5, 20], Data-Type Reduction (DTR) has been shown to be a valuable abstraction for system with dynamic process creation and destruction, and in [2] we already sketched how DTR applies to the special case of DCS models.

DTR belongs to the category of *heterogeneous* abstraction techniques [22] which allow different parts of the system to be abstracted using different degrees of precision. The characteristic of DTR is that it maintains the *full degree of precision* for a fixed and finite set of processes, and *completely* dismisses information of any other processes. That is, DTR follows spotlight principle [19] by focusing on a finite set of processes and abstracting the rest into one *summary process* that represents the behaviour of *any* unfocused process.

The advantage of employing these two extrema of precision is that the abstract transition relation can be easily computed by a syntactical transformation

of the system [5]. This is in contrast to homogeneous abstraction techniques like predicate abstraction [16] where the computation of the abstraction itself is the performance bottleneck. However, the disadvantage of entirely abstracting the behaviour of unfocused processes is a large amount of spurious behaviour, i.e. runs in the abstracted system that are not present in the original system. The spurious behaviour stems from invalid interferences of abstracted processes with concrete processes. [5] suggests to exclude these unwanted interferences by adding so-called non-interference lemmata which are of the form

*“If some process sends something to me, then it is allowed to do so.”*

It is however not obvious whether and how concrete instances of this pattern can automatically be inferred from the system. Furthermore, once they are found, the validity of these lemmata has to be proven to be correct for the original (and thus infinite state) system as additional verification tasks.

In this paper, we devise a technique to automatically generate and integrate non-interference properties into the DTR abstraction. For this purpose, we keep track of *valid* interferences of the summary process. Roughly spoken, for each message  $m$  and concrete process  $\iota$ , we count the number of summarised processes that currently may send  $m$  to  $\iota$ . To keep this information finite, we count up to a finite number and fall back to uncertainty only if this number is exceeded, i.e. we apply a variant of Counter Abstraction [11] to that part of the system that normally is completely blurred in the DTR approach.

The process counters themselves are updated based on the communication between concrete processes and the summary process. The reason why this kind of communication observation gives enough information in order to precisely update the process counter is that a communicating system usually exhibits a causal relationship among the messages, e.g. if a car sends a *split* command to me, then I have requested a *merge* from it beforehand, and in between no *split* commands have been exchanged. Thus, a concrete car sending a *merge* message to the summary process will increase the counter for valid *split* interferences from the summary, and the reception of a *split* from the summary process will decrease this counter again.

This paper makes two contributions. Firstly, as a continuation of our work in [2], we provide a formal definition of DTR for DCS models. Secondly, our main contribution is a method to automatically generate and integrate non-interference properties into the abstraction by combining Data-Type Reduction with a variant of Counter Abstraction.

*Structure.* Section 2 introduces a concise modelling language for Dynamic Communication Systems (DCS), based on the concepts defined in [2]. Section 3 explains our approach for analysing DCS models in detail. We start in subsection 3.1 by formally defining the technique of Data-Type Reduction for DCS models, and motivate the need for further refinement of the abstraction. Subsection 3.2 then describes our method to automatically obtain this kind of refinement. Section 4 discusses related work, and Sect. 5 concludes.

## 2 Dynamic Communication Systems

In [2], we introduce *DCS protocols* as a modelling language for *Dynamic Communication Systems*, basically as an extension of Communicating Finite State Machines [3] by dynamic creation of processes and dynamic topologies. Thereby, DCS provides for an alternative to the well-known  $\pi$ -calculus [13] in which the important language constructs of DCS only appear in an encoded form and are not directly accessible for a tailored analysis (cf. [2]). In this paper, we define a restricted synchronous variant of DCS protocols that still contains the relevant features for both adequately modelling DCS and explaining our approach.

*DCS Overview.* A global DCS state, called topology, comprises a set of processes. Each process is described by its unique identity and its configuration, that is, its local state and its links to other processes. A topology can be extended by adding a process in an initial configuration to it. On the other hand, a process can disappear from a topology if it is in a fragile configuration (see below).

The behaviour of each of the processes is defined in terms of a *DCS protocol* and each process operates on a sole copy of this protocol description, similarly to instances of classes in object-oriented modelling. A DCS protocol defines the set of *states* a process can assume, the set of *channels* by which a process can link itself to other processes and the set of *messages* a process can send and receive. Transitions among the states are annotated by sending and reception of messages over channels. A sent message always includes the identity of the sender process, and the receiver process can store the attached identity into one of its channels. This is the mechanism to establish links between processes. To remove an existing link to another process, each transition has a boolean *reset flag* indicating whether the corresponding channel is cleared after taking this transition. To bootstrap the linking procedure as sketched above, a process can receive a so-called *external message* that carries some identity from the set of currently existing processes.

**Definition 1 (DCS Protocol).** *A DCS Protocol is a tuple*

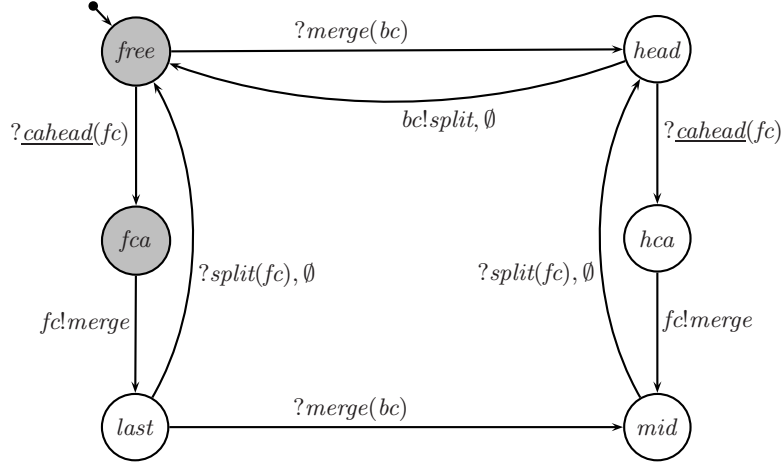
$$P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$$

where

- $Q$  is a finite set of states,
- $A \subseteq Q$  is the set of initial states,
- $\Omega \subseteq Q$  is the set of fragile states,
- $\chi$  is a finite set of channel names,
- $\Sigma$  is a finite set of message names,
- $\Sigma_X \subseteq \Sigma$  is the set of external message names, and
- $succ \subseteq Q \times \chi \times \{!, ?\} \times \Sigma \times \mathbb{B} \times Q$  is the successor relation. ◇

For a transition  $tr = (q, c, sr, m, r, q') \in succ$ , we use  $q(tr)$ ,  $c(tr)$ ,  $sr(tr)$ ,  $m(tr)$ ,  $r(tr)$ , and  $q'(tr)$  to denote the respective components of  $tr$ . The transition  $tr$  is

called a send transition if  $sr = !$  and a receive transition if  $sr = ?$ . The effect of a send transition is to send the message  $m$  to the process stored in channel  $c$ . The effect of a receive transition is to store the identity of the sender of message  $m$  in channel  $c$ . In both cases, the process then moves from state  $q$  to  $q'$  and clears the channel  $c$  iff the reset flag  $r$  is *true* (cf. Def. 2 below for a formal definition).



**Fig. 2. DCS Protocol for platoon merge and split.** For channel  $c$  and message  $m$ , a send transition is written in the form ' $c!m$ ' and a receive transition as ' $?m(c)$ '. If the reset flag  $r$  is set for a transition, this is indicated by the ' $\emptyset$ ' symbol.

*The Running Example.* Figure 2 shows the DCS protocol 'platoon' implementing the merge and split manoeuvres. It comprises six states, two channels and three messages where *free* is the single initial state and the (gray marked) states *free* and *fca* are fragile states. The (underlined) message *cahead* is an external message.

The *merge manoeuvre* starts when a process in state *free* (a car not involved in a platoon) or in state *head* (the head of a platoon) receives the external message *cahead* (car ahead). It stores the attached identity in its channel *fc* (front car). The process then proceeds to state *last* (the last car in a platoon) resp. state *mid* (in the middle of a platoon) if it can synchronise with the process denoted by its *fc* channel on message *merge*. The process receiving the *merge* message proceeds from state *free* to *head* resp. from *last* to *mid*, thereby storing the received identity in its channel *bc* (back car). Repeating these protocol steps allows to build platoons of arbitrary lengths.

The *split manoeuvre* is initiated the head of the platoon by synchronising on *split* with its back car on the transition from *head* to *free*. It thereby removes the process from its *bc*-channel. The back car clears its *fc*-channel on reception

of *split*. If it is in state *last*, it moves to state *free* and the platoon is completely split. If it is in state *mid*, it becomes the new head of the platoon and itself initiates the splitting of the remaining platoon.

*DCS Configurations.* For the rest of this paper, let  $Id$  be a countably infinite set of process identities. Given a DCS protocol  $P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$ , a configuration of  $P$  is a tuple

$$(q, C)$$

where  $q \in Q$  is a state and  $C : \chi \rightarrow Id$  is a partial evaluation of the channels. A configuration is called initial if  $q \in A$  and  $C(c)$  is undefined for all  $c \in \chi$ . A configuration is called fragile if  $q \in \Omega$ . The set of all configurations of  $P$  is denoted by  $\mathcal{L}(P)$ .

Having defined a (local) configuration of a DCS protocol, a global configuration, called topology, of  $P$  is a partial function  $T : Id \rightarrow \mathcal{L}(P)$ . The idea is that the domain of  $T$ , written  $\text{dom}(T)$ , describes the set of processes existing in  $T$ , and  $T(\iota)$  yields the (local) configuration of each  $\iota \in \text{dom}(T)$ . The set of all topologies of  $P$  is denoted by  $\mathcal{T}_{Id}(P)$ .

*The Running Example.* The snapshot presented in Fig. 1 corresponds to the following topology

$$\begin{aligned} T_S = [\iota_1 \mapsto (free, []), \iota_2 \mapsto (last, [fc \mapsto \iota_3]), \\ \iota_3 \mapsto (mid, [fc \mapsto \iota_4, bc \mapsto \iota_2]), \iota_4 \mapsto (head, [bc \mapsto \iota_3])] \end{aligned} \quad (1)$$

with  $\text{dom}(T_S) = \{\iota_1, \iota_2, \iota_3, \iota_4\} \subset Id$ .

*DCS Semantics.* We now specify under which conditions a DCS topology can evolve. As already sketched, a topology can be extended by a new process, an existing process can disappear, a process can receive an external message and processes can communicate among each other.

**Definition 2 (Topology Evolution).** Let  $P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$  be a DCS protocol. Two topologies  $T, T' \in \mathcal{T}_{Id}(P)$  evolve, written  $T \rightarrow T'$ , if exactly one of the following four conditions is satisfied:

**Process Appearance (PA)** A process  $\iota \in Id$  freshly appears, i.e.  $\text{dom}(T') = \text{dom}(T) \dot{\cup} \{\iota\}$  and  $T'(\iota)$  is initial.

**External Message (EM)** A process  $\iota \in \text{dom}(T)$  in configuration  $T(\iota) = (q, C)$  receives an external message on transition  $(q, c, ?, m, r, q') \in succ$  with  $m \in \Sigma_X$ , i.e.  $T'(\iota) = (q', C')$  with

$$C' = \begin{cases} C|_{\text{dom}(C) \setminus \{c\}} & \text{if } r = \text{true} \\ C[c \mapsto \iota'] & \text{if } r = \text{false} \end{cases}$$

for some  $\iota' \in \text{dom}(T)$  with  $\iota' \neq \iota$ .

**Process Synchronisation (PS)** Two processes  $\iota_s, \iota_r \in \text{dom}(T)$  with  $T(\iota_s) = (q_s, C_s)$  and  $T(\iota_r) = (q_r, C_r)$  with  $C_s(c_s) = \iota_r$  synchronise on transitions  $(q_s, c_s, !, m, r_s, q'_s), (q_r, c_r, ?, m, r_r, q'_r) \in \text{succ}$ , i.e.  $T'(\iota_s) = (q'_s, C'_s)$  and  $T'(\iota_r) = (q'_r, C'_r)$  with

$$C'_s = \begin{cases} C_s|_{\text{dom}(C_s)\setminus\{c_s\}} & \text{if } r_s = \text{true} \\ C_s & \text{if } r_s = \text{false} \end{cases}$$

and

$$C'_r = \begin{cases} C_r|_{\text{dom}(C_r)\setminus\{c_r\}} & \text{if } r_r = \text{true} \\ C_r[c_r \mapsto \iota_s] & \text{if } r_r = \text{false}. \end{cases}$$

**Process Disappearance (PD)** A process  $\iota \in \text{dom}(T)$  disappears, i.e.  $T' = T|_{\text{dom}(T)\setminus\{\iota\}}$  and  $T(\iota)$  is fragile.

In each of the four cases, all processes not involved in the current topology evolution are required to remain the same.  $\diamond$

A sequence of topologies  $(T_i)_{i \in \mathbb{N}_0}$  with  $T_i \rightarrow T_{i+1}$  for all  $i \in \mathbb{N}_0$  and  $T_0$  being the initial topology, i.e.  $\text{dom}(T_0) = \emptyset$ , is called a run of P. The semantics of P, denoted  $\llbracket \text{P} \rrbracket_{Id}$ , is the set of all runs of P. We explicitly use the subscript  $Id$  to denote the concrete semantics which employs an *infinite* set of available identities in contrast to modifications of the semantics in later sections.

*The Running Example.* In Fig. 3, we exemplarily sketch a run of the DCS protocol platoon from Fig. 2. Starting at the initial topology, two processes enter the scene. The first one then receives an external message carrying the identity of the second one. Both then agree on the merge transition and become a platoon of size two. The head of the platoon then decides to split again, and disappears from the scene afterwards.

### 3 Analysis of Dynamic Communication Systems

We are interested in the formal verification of properties of DCS that can be expressed as universally quantified first-order formulae in prenex form, i.e.

$$\forall p_1, \dots, p_n . \phi$$

where  $\phi$  is some temporal formula using variables  $p_1$  to  $p_n$ , but with no further quantification. For example, a desirable property of the platooning system is

$$\forall p_1, p_2 . \text{G} (p_1.bc=p_2 \rightarrow p_2.fc=p_1) \quad (2)$$

that is, for all two processes  $p_1$  and  $p_2$ , it is always the case that when process  $p_1$  has  $p_2$  as its back car, then  $p_2$  has  $p_1$  as its front car. For lack of space, we have to refer the reader to [2] for a detailed description of the DCS property language called METT.

$T_0$ (PA)		
$T_1$ (PA)	$\iota_1 \mapsto (free, [])$	
$T_2$ (EM)	$\iota_1 \mapsto (free, []),$ $\iota_2 \mapsto (free, [])$	
$T_3$ (PS)	$\iota_1 \mapsto (fca, [fc \mapsto \iota_2]),$ $\iota_2 \mapsto (free, [])$	
$T_4$ (PS)	$\iota_1 \mapsto (last, [fc \mapsto \iota_2]),$ $\iota_2 \mapsto (head, [bc \mapsto \iota_1])$	
$T_5$ (PD)	$\iota_1 \mapsto (free, []),$ $\iota_2 \mapsto (free, [])$	
$T_6$	$\iota_1 \mapsto (free, [])$	

**Fig. 3. Concrete run of platoon.** The first column names the topology and the kind of evolution (cf. Def. 2) to the next topology. The second column shows the topology in a formal notation and the third column visualises it in a graph notation.

However, formal analysis of DCS protocols faces the problem that the semantics of a DCS protocol is an infinite set of runs, due to the infinite set of available identities. A straight-forward approach to enable automated verification is the restriction to a *finite* subset of identities. In general, considering only a subset of identities  $Id' \subseteq Id$  for a DCS protocol  $P$  means that the *Process Appearance* condition in Def. 2 only picks identities from  $Id'$ . We thereby obtain the underapproximated semantics, denoted  $\llbracket P \rrbracket_{Id'}$ . As every run with identities  $Id'$  is also a run with identities  $Id$ , we have

$$\llbracket P \rrbracket_{Id'} \subseteq \llbracket P \rrbracket_{Id}$$

for any  $Id' \subseteq Id$  (especially if  $Id'$  is a *finite* subset of  $Id$ ). Thus the technique of underapproximation is good for finding errors in the protocol by limiting the analysis to a finite set of processes, e.g. check if two cars in isolation adhere to property (2). However, as any influences from processes from  $Id \setminus Id'$  are disregarded, the absence of unwanted behaviour in general can not be guaranteed by considering only finitely many processes. For example, to show that a car platoon behaves correctly under any influence of arbitrarily many other cars, we need something stronger.

### 3.1 Data-Type Reduction for DCS

Data-Type Reduction yields a finite state representation of an infinite state system by applying the spotlight principle [19], that is, focus on a finite set of processes and represent these processes precisely. Any information about the rest, i.e. the processes “in the shadows”, is completely dismissed. To provide a sound overapproximation of the system, a special identity  $\perp$  is introduced to

summarise the behaviour of any process that is not in the focus of the spotlight. Concrete processes will still be able to communicate with  $\perp$ , but no information about  $\perp$  will be stored. DTR was originally introduced for the verification of parameterised systems [12], and has been demonstrated to be suitable for systems with unbounded dynamic creation and destruction in [5]. In the following, we formally describe how DTR applies to Dynamic Communication Systems as defined in the previous section.

*Abstract DCS Configurations.* For the rest of the paper, let  $Id' \subset Id$  be a finite subset of identities. Given a protocol  $P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$ , an *abstract configuration* of  $P$  is a tuple  $(q, C^\sharp)$  where  $q \in Q$  is a state and  $C^\sharp : \chi \rightarrow Id' \cup \{\perp\}$  is a partial evaluation of the channels. The set of all abstract configurations of  $P$  is denoted by  $\mathcal{L}^\sharp(P)$ . An abstract topology of  $P$  is a partial function  $T^\sharp : Id' \rightarrow \mathcal{L}^\sharp(P)$ , and the set of all abstract topologies of  $P$  is denoted by  $\mathcal{T}_{Id'}^\sharp(P)$ .

The DTR abstraction of a topology  $T \in \mathcal{T}_{Id}(P)$  with respect to  $Id'$ , denoted  $\alpha_{Id'}(T)$ , is basically a restriction of the domain of  $T$  to  $Id'$ . Additionally, all remaining local configurations  $(q, C)$  in  $\text{ran}(T|_{Id'})$  are modified such that each channel  $c \in \chi$  with  $C(c) \in Id \setminus Id'$  is set to  $C(c) = \perp$ .

*The Running Example.* The DTR abstraction of the topology (1) of Fig. 1 with respect to  $\{\iota_1, \iota_2, \iota_3\}$  yields the abstract topology

$$[\iota_1 \mapsto (free, []), \iota_2 \mapsto (last, [fc \mapsto \iota_3]), \iota_3 \mapsto (mid, [fc \mapsto \perp, bc \mapsto \iota_2])], \quad (3)$$

that is,  $\iota_4$  has disappeared from the domain and the  $fc$  link of  $\iota_3$  is set to  $\perp$ .

*DTR Semantics of DCS.* The evolution of abstract topologies is based on the evolution of concrete topologies as defined in Def. 2 as follows. Firstly, two modifications are to be made, namely that *Process Appearance* only creates processes from the finite set  $Id'$ , and *External Messages* can also carry the abstract identity  $\perp$ . Secondly, the possibility to synchronise a concrete process with  $\perp$  is added, such that messages can be sent to and received from the abstract process.

**Definition 3 (DTR Topology Evolution).** Let  $P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$  be a DCS protocol. Two topologies  $T_1^\sharp, T_2^\sharp \in \mathcal{T}_{Id'}^\sharp(P)$  evolve under DTR, written  $T_1^\sharp \rightsquigarrow T_2^\sharp$ , if  $T_1^\sharp = T_2^\sharp$ , or  $T_1^\sharp \rightarrow T_2^\sharp$  by (PS) or (PD) as defined in Def. 2, or if exactly one of the following conditions is satisfied:

**Process Appearance (PA)** A process  $\iota \in Id'$  freshly appears, i.e.  $\text{dom}(T_2^\sharp) = \text{dom}(T_1^\sharp) \cup \{\iota\}$  and  $T_2^\sharp(\iota)$  is initial.

**External Message (EM)** A process  $\iota \in \text{dom}(T_1^\sharp)$  in configuration  $T_1^\sharp(\iota) = (q, C_1^\sharp)$  receives an external message on transition  $(q, c, ?, m, r, q') \in succ$  with  $m \in \Sigma_X$ , i.e.  $e T_2^\sharp(\iota) = (q', C_2^\sharp)$  with

$$C_2^\sharp = \begin{cases} C_1^\sharp|_{\text{dom}(C_1^\sharp) \setminus \{c\}} & \text{if } r = \text{true} \\ C_1^\sharp[c \mapsto \iota'] & \text{if } r = \text{false} \end{cases}$$

for some  $\iota' \in \text{dom}(T_1^\sharp) \cup \{\perp\}$  with  $\iota' \neq \iota$ .

**Send to Summary (SS)** A process  $\iota \in \text{dom}(T_1^\sharp)$  in configuration  $T_1^\sharp(\iota) = (q_s, C_1^\sharp)$  with  $C_1^\sharp(c_s) = \perp$  synchronises with  $\perp$  on transitions

$$\begin{aligned} tr_s &= (q_s, c_s, !, m, r_s, q'_s) \in \text{succ} \\ tr_r &= (q_r, c_r, ?, m, r_r, q'_r) \in \text{succ} \end{aligned}$$

i.e.  $T_2^\sharp(\iota) = (q'_s, C_2^\sharp)$  with

$$C_2^\sharp = \begin{cases} C_1^\sharp|_{\text{dom}(C_1^\sharp) \setminus \{c_s\}} & \text{if } r_s = \text{true} \\ C_1^\sharp & \text{if } r_s = \text{false}. \end{cases}$$

**Receive from Summary (RS)** A process  $\iota \in \text{dom}(T_1^\sharp)$  in configuration  $T_1^\sharp(\iota) = (q_r, C_1^\sharp)$  synchronises with  $\perp$  on transitions

$$\begin{aligned} tr_s &= (q_s, c_s, !, m, r_s, q'_s) \in \text{succ} \\ tr_r &= (q_r, c_r, ?, m, r_r, q'_r) \in \text{succ} \end{aligned}$$

i.e.  $T_2^\sharp(\iota) = (q'_r, C_2^\sharp)$  with

$$C_2^\sharp = \begin{cases} C_1^\sharp|_{\text{dom}(C_1^\sharp) \setminus \{c_r\}} & \text{if } r_r = \text{true} \\ C_1^\sharp[c_r \mapsto \perp] & \text{if } r_r = \text{false}. \end{cases}$$




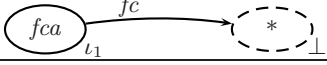

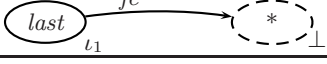



In each case, all processes not involved in the current topology evolution are required to remain the same.  $\diamond$

A sequence of abstract topologies  $(T_i^\sharp)_{i \in \mathbb{N}_0}$  with  $T_i^\sharp \rightsquigarrow T_{i+1}^\sharp$  for all  $i \in \mathbb{N}_0$  and  $T_0^\sharp$  being the initial topology, i.e.  $T_0^\sharp = T_0$ , is called an abstract run of  $\mathsf{P}$ . The abstract semantics of  $\mathsf{P}$ , denoted  $\llbracket \mathsf{P} \rrbracket_{Id'}^\sharp$ , is the set of all abstract runs of  $\mathsf{P}$ .

*The Running Example.* An abstract run of platoon with a single identity  $\iota_1$  is shown in Fig. 4. Clearly, the underapproximated semantics with a single identity does not comprise a topology where the single process reaches the state *last* as there is no other process to merge with.

Note that Def. 3.(RS) reveals the very coarse abstraction employed by the DTR approach, as no constraints about the relation between  $\perp$  and the receiver process  $\iota$  are required. The summary process  $\perp$  can send any message, as long there is a corresponding sending transition  $tr_s$  *somewhere* in the DCS protocol. However, the benefit of DTR is that computing the abstract system is as easy as computing the underapproximated system, as Def. 3 only considers the local information of finitely many processes.

DTR provides a sound abstraction for DCS as follows. The abstraction of a concrete run  $\sigma = (T_i)_{i \in \mathbb{N}_0} \in \llbracket \mathsf{P} \rrbracket_{Id}$ , denoted  $\alpha_{Id'}(\sigma)$ , is the sequence of abstracted topologies  $(\alpha_{Id'}(T_i))_{i \in \mathbb{N}_0}$ . Then we have:

$T_0^\sharp$ (PA)			
$T_1^\sharp$ (EM)	$\iota_1 \mapsto (free, [])$		
$T_2^\sharp$ (SS)	$\iota_1 \mapsto (fca, [fc \mapsto \perp])$		
$T_3^\sharp$ (RS)	$\iota_1 \mapsto (last, [fc \mapsto \perp])$		
$T_4^\sharp$	$\iota_1 \mapsto (free, [])$		

**Fig. 4. Abstract run of platoon.** The first column names the topology and the kind of evolution (cf. Def. 3) to the next topology. The second column shows the topology in a formal notation and the third column visualises it in a graph notation.

**Theorem 1 (Soundness of DTR abstraction).** *Let  $\mathsf{P}$  be a DCS protocol. The abstraction of each concrete run of  $\mathsf{P}$  is an abstract run of  $\mathsf{P}$ , i.e.*

$$\sigma \in \llbracket \mathsf{P} \rrbracket_{Id} \implies \alpha_{Id'}(\sigma) \in \llbracket \mathsf{P} \rrbracket_{Id'}^\sharp \quad \diamond$$

**Proof.** For any  $T_1, T_2 \in \mathcal{T}_{Id}(\mathsf{P})$ , we can show  $T_1 \rightarrow T_2 \implies \alpha_{Id'}(T_1) \rightsquigarrow \alpha_{Id'}(T_2)$  by close examination of the four cases from Def. 2. It is straight-forward to see that the abstract evolution to  $\alpha_{Id'}(T_2)$  according to Def. 3 preserves the same information concerning the summary process  $\perp$  as the abstraction of the concrete topology  $T_2$  does, namely links to  $\perp$  but no information about  $\perp$  itself.  $\square$

Theorem 1 states any behaviour of a finite subset of identities  $Id'$  that is possible in the concrete semantics, is also possible in the abstract semantics. Note that this kind of behaviour preservation is best suited to analyse universally quantified DCS properties of the form  $\forall p_1, \dots, p_n \cdot \phi$ , as these properties express relationships among *finitely many* processes. In fact, the heuristic of the DTR approach to determine the set of concretely represented processes is to choose  $Id' \subset Id$  such that  $|Id'| = n$ . If  $\phi$  can be proven correct for a concrete binding of  $p_i$  to  $Id'$ , symmetry arguments allow to conclude the correctness of the whole first-order formula for the original system. This concluding step is known under the term Query Reduction [21].

However, DTR abstraction is not complete, i.e. in general there are abstract runs  $\sigma^\sharp$  in  $\llbracket \mathsf{P} \rrbracket_{Id'}^\sharp$  for which no concrete run  $\sigma \in \llbracket \mathsf{P} \rrbracket_{Id}$  with  $\alpha_{Id'}(\sigma) = \sigma^\sharp$  exists. Moreover, the maximal coarse abstraction of unfocused processes allows for a *large* amount of spurious behaviour, that in practice often prevents to successfully prove relevant properties of the systems via DTR (cf. [2]).

*The Running Example.* To see an example of such spurious behaviour, consider the topology  $T_4$  as shown in Fig. 3. The abstract evolution according to Def. 3 allows the abstract process to send a *split* message to process  $\iota_1$ , i.e. the topology

$$[\iota_1 \mapsto (last, [fc \mapsto \iota_2]), \iota_2 \mapsto (head, [bc \mapsto \iota_1])] \quad (4)$$

may evolve under DTR by Def. 3.(RS) to the topology

$$[\iota_1 \mapsto (\text{free}, []), \iota_2 \mapsto (\text{head}, [bc \mapsto \iota_1])], \quad (5)$$

that is,  $\iota_1$  has “stealthily” split from his front car  $\iota_2$ . Note that we thereby have reached a topology violating property (2), but we do not a priori know whether we encountered spurious or valid behaviour in order to reach this topology. By *manually* investigation of the platoon protocol, we can identify the interference of the summary as *spurious* as only the front car is supposed to initiate the splitting from its back car. In the setting above however, some car completely unrelated to car  $\iota_1$  has requested the split.

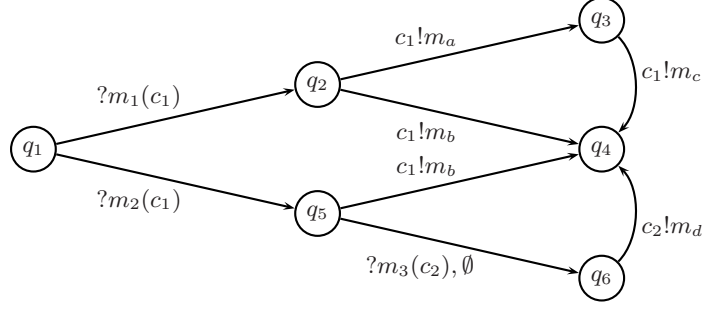
*Wrapping Up.* Intuitively, DTR turns the falsification capabilities of the underapproximated semantics into a proper verification techniques by putting the isolated finite set of identities into an environment that summarises an arbitrary number of other processes. However, this summary process shows in general more behaviour than the processes it summarises. Thus to be effective for verification, DTR abstraction needs a sound refinement of the summary process’ behaviour. As sketched in the introduction, the integration of non-interference lemmata gives a good intuition of “what to do”, but it does not provide a method of “how to do so”. In the next section, we show how to automatically generate non-interference properties that effectively eliminate spurious behaviour from DTR abstractions, thus turning the manual investigation sketched above into an automatic procedure.

### 3.2 Refining Data-Type Reduction by Process Counting

The spurious example from the last section shows that the DTR abstraction allows for message interferences of completely unrelated processes. Intuitively, the reason for this artifact of the abstraction is that no information about the relation from summarised processes to concrete processes, i.e. no links from  $\perp$  to processes from  $Id'$ , is maintained.

However, a characteristic of DCS is that links are established via message communication, thus the validity of a message interference from  $\perp$  to some  $\iota' \in Id'$  depends on prior communication among  $\iota'$  and  $\perp$ . We exploit this fact by maintaining information about the summary process in terms of messages that have been “enabled” by prior communication between  $\perp$  and processes from  $Id'$ .

For this, note that we can (statically) derive from a DCS protocol which messages are next to be send over some channel  $c$  if a process is in state  $q$ . Consider the DCS example depicted in Fig. 5. The next messages a process in state  $q_2$  can send over channel  $c_1$  are  $m_a$  and  $m_b$ . From state  $q_5$ , the next enabled message for channel  $c_1$  is only  $m_b$ . Thus a concrete process  $\iota'$  sending  $m_1$  to  $\perp$  enables the set  $\{m_a, m_b\}$ , and sending  $m_2$  enable the set  $\{m_b\}$ . Note that both sets can be enabled in one topology (although one process cannot be in state  $q_2$  and  $q_5$  simultaneously) as  $\perp$  represents *multiple* processes. In addition to communication from  $\iota'$  to  $\perp$ , also  $\perp$  sending a message that has been enabled



**Fig. 5. Artificial DCS Protocol** to demonstrate the concept of enabled messages.

before alters the set of enabled messages. Clearly, sending back  $m_a$  to  $\iota'$  will *disable* the complete set  $\{m_a, m_b\}$ . The consequence of sending back  $m_b$  to  $\iota'$  is more difficult to see, as we cannot decide whether this particular sending of  $m_b$  has been enabled by itself upon reception of  $m_2$ , or together with  $m_a$  by reception of  $m_1$ . Thus both alternatives have to be considered, i.e. one possible update of enabled message is to disable the set  $\{m_b\}$ , the other to disable the set  $\{m_a, m_b\}$ . Sending messages from  $\perp$  to  $Id$  may also *enable* new message sending, e.g. sending message  $m_a$  will enable the set  $\{m_c\}$ . As a final remark, note that sending  $m_3$  to  $\perp$  must not enable  $\{m_d\}$  as the channel  $c_2$  is cleared upon reception of  $m_3$ .

Unfortunately, we cannot maintain the complete information of enabled message sets as described above, as one set can be enabled many times at once. For example, process  $\iota'$  sending  $m_2$  twice enables the set  $\{m_b\}$  twice, namely for two different processes represented by  $\perp$ . A natural solution to avoid infinite counting is to do finite counting up to some  $K$  and fall back to uncertainty if the counter exceeds  $K$ , i.e. to apply Counter Abstraction [11].

*Formalising the approach.* In the following, let  $P = (Q, A, \Omega, \chi, \Sigma, \Sigma_X, succ)$  be a DCS protocol. A sequence of transitions  $tr_0 tr_1 \dots tr_n$  is called a transition path of  $P$  if  $q'(tr_i) = q(tr_{i+1})$  for all  $0 \leq i < n$ . The set of all paths of  $P$  is denoted by  $Paths(P)$ . The set  $E_P(q, c)$  determines the set of messages which are the next to be send over channel  $c$  if a process is in state  $q$ , according to  $succ$ .

$$E_P(q, c) = \{m \in Msgs \mid \exists tr_0 tr_1 \dots tr_n \in Paths(P) : q(tr_0) = q \wedge c(tr_n) = c \wedge m(tr_n) = m \wedge sr(tr_n) = ! \wedge \forall 0 \leq i < n : c(tr_i) \neq c\}$$

For each concretely represented process  $\iota' \in Id'$  and set of messages  $M \subseteq \Sigma$ , we keep track of those processes from  $Id \setminus Id'$  that are able to send some  $m \in M$  to  $\iota'$ . That is, given a concrete topology  $T \in \mathcal{T}_{Id}(P)$ , we define the set of potential communication partners (CP) for  $\iota' \in Id$  and  $M \subseteq \Sigma$  as

$$CP_T(\iota', M) := \{\iota \in Id \setminus Id' \mid T(\iota) = (q, C) \wedge \exists c \in \chi : C(c) = \iota' \wedge M = E_P(q, c)\}.$$

To keep the information finite, we do not keep track of the process identities themselves, but rather we only *count* the number of processes in each set  $CP_T(\iota', M)$  up to a finite number  $K$  as follows. Let  $\mathbb{N}_K := \{0, 1, \dots, K, \infty\}$ . The order  $\leq_K$  is defined on  $\mathbb{N}_K$  as  $n_1 \leq_K n_2$  iff  $n_2 = \infty$  or  $n_1 \leq n_2 \leq K$ . Addition and subtraction in  $\mathbb{N}_K$  are defined as follows:

$$n_1 \oplus_K n_2 := \begin{cases} \infty & \text{if } n_1 = \infty \vee n_2 = \infty \vee n_1 + n_2 > K \\ n_1 + n_2 & \text{else} \end{cases}$$

$$n_1 \ominus_K n_2 := \begin{cases} \infty & \text{if } n_1 = \infty \\ 0 & \text{if } n_1 \neq \infty \wedge n_2 = \infty \vee n_1 - n_2 < 0 \\ n_1 - n_2 & \text{else} \end{cases}$$

A *process counter* of  $\mathsf{P}$  assigns a value to a pair of identity and set of messages, i.e. it is a function  $\Pi : (Id \times 2^\Sigma) \rightarrow \mathbb{N}_K$ . The set of all process counters of  $\mathsf{P}$  is denoted  $\mathcal{PC}(\mathsf{P})$ . Given two process counters  $\Pi_1, \Pi_2 \in \mathcal{PC}(\mathsf{P})$ , we say that the counter  $\Pi_1$  is smaller than  $\Pi_2$ , written  $\Pi_1 \preceq_K \Pi_2$ , if  $\Pi_1((\iota, M)) \leq_K \Pi_2((\iota, M))$  for all  $\iota \in Id'$  and  $M \subseteq \Sigma$ . The addition of  $\Pi_1$  and  $\Pi_2$ , denoted  $\Pi_1 \oplus_K \Pi_2$ , is defined pointwise as  $(\Pi_1 \oplus_K \Pi_2)((\iota, M)) := \Pi_1((\iota, M)) \oplus_K \Pi_2((\iota, M))$  for all  $\iota \in Id'$  and  $M \subseteq \Sigma$ .

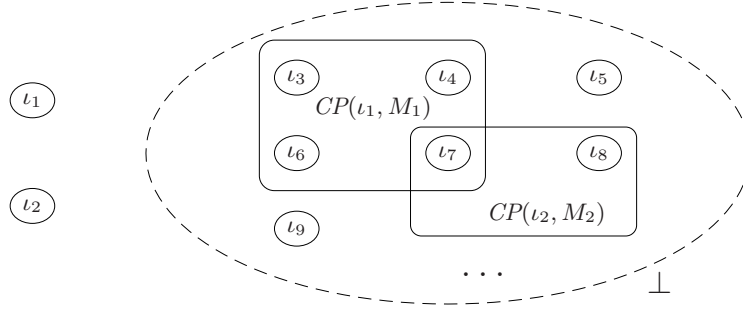
For a given concrete topology  $T \in \mathcal{T}_{Id}(\mathsf{P})$ , we compute the *derived* process counter with respect to  $Id'$ , written  $\pi_{Id'}(T)$ , for  $\iota' \in Id'$  and  $M \subseteq \Sigma$  as

$$\pi_{Id'}(T)((\iota', M)) = \begin{cases} \infty & \text{if } |CP(\iota', M)| > K \\ |CP(\iota', M)| & \text{else.} \end{cases}$$

Figure 6 depicts a topology with two sets of communication partners. We assume that the processes  $\iota_3, \iota_4, \iota_6, \iota_7$  are in a situation that allows them to send some message from  $M_1$  to  $\iota_1$  in the future. For  $\iota_2$ , the processes  $\iota_7$  and  $\iota_8$  may send some message from  $M_2$ . When abstracting this topology, only the process counter will be maintained, e.g. for  $K = 3$  we obtain  $\Pi((\iota_1, M_1)) = \infty$  and  $\Pi((\iota_2, M_2)) = 2$ .

*Using process counters.* The derived process counter represents the most precise information we can get from a topology  $T$  in terms of process counting. Unfortunately, when performing the abstract evolution according to Def. 3 we cannot compute the derived process counter *directly* as we have no information about the configurations of processes from  $Id \setminus Id'$ . Thus the main problem to be solved is how to (precisely) update the process counter based on the information that is present in the abstract topology evolution. As sketched in the beginning of the section, a concretely represented process sending a messages to  $\perp$ , i.e. to some process  $Id \setminus Id'$  in the original system, will enable  $\perp$  to communicate with  $\iota'$  afterwards, as it can store the identity  $Id'$  in one of its channel. That is, the corresponding counters for the enabled messages have to be increased.

On the other hand, a counter for a pair of process  $\iota'$  and set of messages  $M$  has to be decreased after  $\perp$  has sent a message  $m \in M$  to  $\iota'$ . However, if the



**Fig. 6. Communication Partners** in a concrete topology. After abstracting with respect to  $\iota_1$  and  $\iota_2$ , only the process counter will be preserved in order to provide a finite characterisation of the summary process  $\perp$ .

channel is not cleared when sending  $m$ , then other messages may now be enabled which are next to be send according to the DCS protocol.

Maintaining a precise process counter allows to eliminate spurious interferences as  $\perp$  is now only allowed to send message  $m$  to  $\iota'$  if there is a set of messages  $M$  with  $m \in M$  for which the counter of the pair  $\iota'$  and  $M$  is *not zero*.

Formalising the ideas sketched above, we define the refined DTR topology evolution that modifies and respects the status of the process counter as follows.

**Definition 4 (Refined DTR Topology Evolution).** *Let  $P$  be a DCS protocol. Two topologies  $T_1^\sharp, T_2^\sharp \in \mathcal{T}_{Id}^\sharp(P)$  and two process counters  $\Pi_1, \Pi_2 \in \mathcal{PC}(P)$  evolve under DTR, written  $(T_1^\sharp, \Pi_1) \rightsquigarrow (T_2^\sharp, \Pi_2)$ , if  $(T_1^\sharp, \Pi_1) = (T_2^\sharp, \Pi_2)$ , or  $T_1^\sharp \rightsquigarrow T_2^\sharp$  by (PA), (EM), (PS), or (PD) as defined in Def. 3 and  $\Pi_1 = \Pi_2$ , or if exactly one of the following conditions is satisfied:*

**Refined Send to Summary (rSS)** *A process  $\iota \in \text{dom}(T)$  synchronises with  $\perp$  on transitions  $tr_s, tr_r$  as defined in Def. 3.(SS) and*

$$\Pi_2 = \begin{cases} \Pi_1 & \text{if } r(tr_r) = \text{true} \\ \Pi_1[(\iota, E) \mapsto \Pi_1((\iota, E)) \oplus_K 1] & \text{if } r(tr_r) = \text{false} \end{cases}$$

where  $E = E_P(q'(tr_r), c(tr_r))$ .

**Refined Receive from Summary (rRS)** *A process  $\iota \in \text{dom}(T)$  synchronises with  $\perp$  on transitions  $tr_s, tr_r$  as defined in Def. 3.(RS) if there exists a set of messages  $M$  with  $m(tr_s) \in M$  such that*

$$\Pi_1((\iota, M)) \geq 1$$

and

$$\Pi_2 = \begin{cases} \Pi_1[(\iota, M) \mapsto \Pi_1((\iota, M)) \ominus_K 1] & \text{if } r(tr_s) = \text{true} \\ \Pi_1[(\iota, M) \mapsto \Pi_1((\iota, M)) \ominus_K 1][(\iota, E) \mapsto \Pi_1((\iota, E)) \oplus_K 1] & \text{else} \end{cases}$$

where  $E = E_P(q'(tr_s), c(tr_s))$ .

In each case, all processes not involved in the current topology evolution are required to remain the same.  $\diamond$

To explain a run of  $\mathsf{P}$  under refined DTR evolution, we have to specify the initial process counter corresponding to the initial topology. Note that the initial process counter may not be zero for all pairs of processes and messages, although the initial topology does not comprise any processes. The reason is that, at any time, processes summarised by  $\perp$  may receive an external message carrying a process identity from  $\mathcal{I}d'$  and thus may communicate with  $\iota'$  afterwards. Unfortunately, external messages to  $\perp$  are not visible under DTR evolution, thus the initial process counter has to cater for this communication possibilities by enabling an arbitrary number of those messages that follow the reception of external messages in the DCS protocol. In the running example, sending *merge* messages from  $\perp$  to some concrete process  $\iota'$  must always be possible, as a process summarised by  $\perp$  may always receive an external message *cahead* carrying the identity  $\iota'$ . Thus we define the *initial* process counter of  $\mathsf{P}$  as

$$\Pi_0((\iota', M)) := \begin{cases} \infty & \text{if } \exists tr \in succ : m(tr) \in \Sigma_X \wedge r(tr) = false \wedge \\ 0 & \text{else} \quad M = E_{\mathsf{P}}(q'(tr), c(tr)) \end{cases}$$

for all  $\iota' \in \mathcal{I}d'$  and  $M \subseteq \Sigma$ .

A sequence of abstract topologies and observers  $((T_i^\sharp, \Pi_i))_{i \in \mathbb{N}_0}$  with  $(T_i^\sharp, \Pi_i) \rightsquigarrow (T_{i+1}^\sharp, \Pi_{i+1})$  for all  $i \in \mathbb{N}_0$  and  $T_0^\sharp$  being the initial topology, i.e.  $T_0^\sharp = T_0$ , and  $\Pi_0$  being the initial observer, is called a refined abstract run of  $\mathsf{P}$ . The refined abstract semantics of  $\mathsf{P}$ , denoted  $\llbracket \mathsf{P} \rrbracket_{\mathcal{I}d'}^{\sharp R}$ , is the set of all its refined abstract runs.

*The Running Example.* Two examples of process counter updating can be obtained by “replaying” the abstract runs from the previous section under the refined abstract evolution. Figure 7 shows the refined abstract run corresponding to Fig. 4. The important observation is that the (rSS) evolution to topology  $T_3$  increase the counter for the pair  $\iota_1$  and  $\{split\}$  to one, as *split* is in  $E_{\mathsf{P}}(head, bc)$ . The subsequent sending of *split* to  $\iota_1$  is thereby enabled in (rRS). For the spurious run from the last section however, we expect the refinement via process counting to obey the evolution from topology (4) to (5). As up to topology (4) no communication with the summary has taken place, the process counter is still the initial one. Especially the counter for the pair  $\iota_1$  and  $\{split\}$  is zero, thus the summary is not allowed to take this<sup>1</sup> transition to reach topology (5).

We observe from Def. 4 that if the evolution (rRS) is possible with some process counter  $\Pi$ , it is also possible with all process counters  $\Pi'$  with  $\Pi \preceq_K \Pi'$ . This motivates the following lemma.

**Lemma 1.** *Let  $T_1^\sharp, T_2^\sharp \in \mathcal{T}_{\mathcal{I}d'}^\sharp(\mathsf{P})$  and  $\Pi_1, \Pi_2 \in \mathcal{PC}(\mathsf{P})$ . If  $(T_1^\sharp, \Pi_1) \rightsquigarrow (T_2^\sharp, \Pi_2)$ , then for all  $\Pi'_1 \in \mathcal{PC}(\mathsf{P})$  with  $\Pi_1 \preceq_K \Pi'_1$  there exists  $\Pi'_2 \in \mathcal{PC}(\mathsf{P})$  such that  $(T_1^\sharp, \Pi'_1) \rightsquigarrow (T_2^\sharp, \Pi'_2)$  and  $\Pi_2 \preceq_K \Pi'_2$ .  $\diamond$*

<sup>1</sup> Actually, the refinement blocks *all* spurious interferences leading to a violation of (2) such that (2) can be proven to be correct for the platooning system by our method.

$T_0^\sharp$ (PA)		$\Pi_0((\iota_1, \{merge\})) = \infty$
$T_1^\sharp$ (EM)	$\iota_1 \mapsto (free, [])$	$\Pi_1((\iota_1, \{merge\})) = \infty$
$T_2^\sharp$ (rSS)	$\iota_1 \mapsto (fca, [fc \mapsto \perp])$	$\Pi_2((\iota_1, \{merge\})) = \infty$
$T_3^\sharp$ (rRS)	$\iota_1 \mapsto (last, [fc \mapsto \perp])$	$\Pi_3((\iota_1, \{merge\})) = \infty$ $\Pi_3((\iota_1, \{split\})) = 1$
$T_4^\sharp$	$\iota_1 \mapsto (free, [])$	$\Pi_4((\iota_1, \{merge\})) = \infty$

**Fig. 7. Refined abstract run of platoon.** The first column names the topology and the kind of evolution (cf. Def. 4) to the next topology. The second column shows the topology in a formal notation and the third column shows the process counter for those entries that are not zero.

We are now ready to prove the soundness of the refined DTR abstraction. The crucial point is to see that the process counter is updated in such a manner that it does not block any valid message sendings from the summarised process.

**Theorem 2 (Soundness of Refined DTR abstraction).** *Let  $\mathsf{P}$  be a DCS protocol. Each concrete run of  $\mathsf{P}$  has an abstract counterpart in the refined abstract semantics of  $\mathsf{P}$ , i.e. there exist process counters  $\Pi_i$  such that*

$$(T_i)_{i \in \mathbb{N}_0} \in \llbracket \mathsf{P} \rrbracket_{Id} \implies ((\alpha_{Id'}(T_i), \Pi_i))_{i \in \mathbb{N}_0} \in \llbracket \mathsf{P} \rrbracket_{Id'}^{\sharp R} \quad \diamond$$

**Proof.** We show that  $H \subseteq \mathcal{T}_{Id}(\mathsf{P}) \times (\mathcal{T}_{Id'}^\sharp(\mathsf{P}) \times \mathcal{PC}(\mathsf{P}))$  with

$$(T, (T^\sharp, \Pi)) \in H \iff \alpha_{Id'}(T) = T^\sharp \wedge (\pi_{Id'}(T) \oplus_K \Pi_0) \preceq_K \Pi$$

is a simulation relation, that is, for all  $T_1 \in \mathcal{T}_{Id}(\mathsf{P})$  and  $(T_1^\sharp, \Pi_1) \in (\mathcal{T}_{Id'}^\sharp(\mathsf{P}) \times \mathcal{PC}(\mathsf{P}))$  with  $H((T_1, (T_1^\sharp, \Pi_1)))$  we have

$$\begin{aligned} \forall T_2 \in \mathcal{T}_{Id}(\mathsf{P}) : T_1 \rightarrow T_2 \exists (T_2^\sharp, \Pi_2) \in (\mathcal{T}_{Id'}^\sharp(\mathsf{P}) \times \mathcal{PC}(\mathsf{P})) : \\ (T_1^\sharp, \Pi_1) \rightsquigarrow (T_2^\sharp, \Pi_2) \wedge H((T_2, (T_2^\sharp, \Pi_2))). \end{aligned}$$

Let  $T_1, T_2 \in \mathcal{T}_{Id}(\mathsf{P})$  with  $T_1 \rightarrow T_2$ . By lemma 1 we are left to show the existence of a process counter  $\Pi_2 \in \mathcal{PC}(\mathsf{P})$  with  $(\pi_{Id'}(T_2) \oplus_K \Pi_0) \preceq_K \Pi_2$  such that

$$(\alpha_{Id'}(T_1), \pi_{Id'}(T_1) \oplus_K \Pi_0) \rightsquigarrow (\alpha_{Id'}(T_2), \Pi_2).$$

We distinguish between the four cases (PA), (EM), (PS), (PD) of concrete topology evolution of Def. 2. In Def. 3, the refined abstract evolution for the cases (PA), (PD), (EM) do not evaluate the current status of process counter but requires it to remain the same. We thus set  $\Pi_2 = \pi_{Id'}(T_1) \oplus_K \Pi_0$  in these cases. The same applies to the (PS) case when either  $\{\iota_s, \iota_r\} \subseteq Id'$  or  $\{\iota_s, \iota_r\} \subseteq Id \setminus Id'$ , i.e. communication is only among concrete resp. summarised processes. Two cases for (PS) are left:

(1)  $\iota_s \in Id', \iota_r \in Id \setminus Id'$ , i.e. a concrete process sends  $m(tr_s)$  to the summary process. The case for  $r(tr_r) = true$  is trivial as neither a change in the derived process counter nor in the evolution of  $\pi_{Id'}(T_1)$  happens. If  $r(tr_r) = false$ , we set  $\Pi_2 := (\pi_{Id'}(T_1) \oplus_K \Pi_0)[(\iota, E) \mapsto \Pi((\iota, E)) \oplus_K 1]$  with  $E = E_P(q'(tr_r), c(tr_r))$  according to Def. 3.(rSS). In  $T_2$ , the effect of  $\iota_s$  sending the message  $m(tr_s)$  to  $\iota_r$  is that the communication partner sets  $CP_{T_2}(\iota, M)$  with  $m(tr_r) \in M$  may be extended by process  $\iota_r$ . By definition of the derived process counter, this ensures  $(\pi_{Id'}(T_2) \oplus_K \Pi_0) \preceq_K \Pi_2$ .

(2)  $\iota_s \in Id \setminus Id', \iota_r \in Id'$ , i.e. the summary process sends  $m(tr_s)$  to the concrete process  $\iota_r$ . As  $T_1 \rightarrow T_2$ , there exists a message set  $M \subseteq \Sigma$  with  $m(tr_s) \in M$  such that the set of communication partners  $CP_{T_1}(\iota_r, M)$  is not the empty set. Thus  $(\pi_{Id'}(T_1) \oplus_K \Pi_0)((\iota_r, M)) \geq 1$ , and we have  $\alpha_{Id'}(T_1) \rightsquigarrow \alpha_{Id'}(T_2)$ .

For showing the existence of  $\Pi_2$ , we set  $\Pi_2$  according to Def. 3.(rRS). Again, the derived process counter  $\pi_{Id'}(T_2)$  is changed in the same manner as  $\Pi_2$ . Especially the reduction of the process counter for the pair  $\iota_r$  and  $m$  is safe as  $\iota_s$  disappears from the corresponding partner sets  $CP_{T_2}(\iota_r, M)$  for  $m(tr_s) \in M$  when computing the derived process counter for  $T_2$ .

To conclude the proof, we have to show that there exists an abstract topology  $T^\sharp$  and a process counter  $\Pi$  for the initial topology  $T_0$  such that  $(T_0, (T^\sharp, \Pi)) \in H$ . As  $\alpha_{Id'}(T_0) = T_0$  and  $\pi_{Id'}(T_0)((\iota, M)) = 0$  for all  $\iota \in Id'$  and  $M \subseteq \Sigma$ , we set  $T^\sharp = T_0$  and  $\Pi = \Pi_0$ . This setting implies  $H(T_0, (T^\sharp, \Pi))$ .  $\square$

*Tool support.* Note that the integration of process counting into the DTR abstraction can be conducted fully automatically. For a given DCS protocol  $P$ , we calculate the sets  $E_P(q, c)$  and adjust the abstract transition relation according to Def. 4. Recently, we have extended the DCS verification framework of Rakow [15] by our method and were able to establish properties like (2) for the car platooning example using standard tools like the SPIN model-checker [8].

*Assessment.* This section has shown that the observation of communication sequences allows us to derive information about the existence of communication partners in the abstracted part of the system. By our method, we are able to automatically establish properties that DTR alone is not able to prove.

However, the demonstrated approach is not able to eliminate all spurious message interferences. Firstly, the sets of communication partners only denote a *potential* communication. It is possible that a message sending that is enabled by prior communication is in fact not executable in the original system, e.g. if the process always gets blocked before. Nevertheless, the summary process is always allowed to send a message that has been enabled. Secondly, the process counter falls back to uncertainty if the counter exceeds  $K$ . The scenario is possible when there is a source of unrestricted sendings in the system, e.g. a concrete process sending perpetually messages without waiting for any answers. We however expect that such kind of polling only occurs in the low-level physical layer, but not on the high-level negotiation protocols addressed by DCS protocols. We thus conjecture that counting up to  $K = 1$  is already sufficient for reasonable DCS protocols.

## 4 Related Work

In the literature, analyses of systems that comprise an unbounded number of processes are mainly developed in terms of a parameterised system  $S(N)$ , that is, a parallel composition of  $N$  identical finite processes. The task is to prove properties of  $S(N)$  for every  $N > 1$ . Various verification approaches have been proposed, e.g. based on network invariants [10], counter abstraction [14], environment abstraction [4], or symmetry reduction [7]. However, parametrised systems have no means of changing the connection topology and basically communicate via a global shared memory. The problem of a changing link structures is mainly addressed in the area of shape analysis [17]. However, the links are only among *data cells*, i.e. the connected nodes do not exhibit a local behaviour. Dynamic Communication Systems (DCS) combine the two difficulties of unbounded processes and dynamic connections and hence require new verification techniques. The verification of Java programs with dynamic thread creation is addressed in [23], based on a 3-valued logic framework. This approach is restricted to safety properties, while Data-Type Reduction is able to preserve liveness properties on the concrete part of the system. We have sketched the verification of DCS already in [2] by combining Data-Type Reduction with global system invariants that denote all possible communication topologies. These invariants are computed by a new form of abstract interpretation of graph transformation rules [1]. This approach is currently worked out in more detail. The ideas presented in this paper present an alternative approach and we have to investigate how both relate to each other in terms of effectiveness and performance.

## 5 Conclusion

Reducing the number of message interferences substantially increases the applicability of Data-Type Reduction for communicating systems of systems. We demonstrate that the observation of communication sequences allows us to obtain valuable information about valid communication partners that is normally dismissed by the abstraction. We thereby extend the idea of manually inferring and adding non-interference lemmata for DTR refinement to a fully automatic procedure. The extension itself is based on a variant of Counter Abstraction.

Future work has to evaluate the approach in terms of effectiveness and performance for larger systems. We furthermore work on a formal characterisation of the “degree of improvement” of DTR by our method, and identify subclasses of DCS where the refinement leads to a *complete* abstraction technique.

## References

1. Jörg Bauer. *Analysis of Communication Topologies by Partner Abstraction*. PhD thesis, Universität des Saarlandes, 2006.
2. Jörg Bauer, Ina Schaefer, Tobe Toben, and Bernd Westphal. Specification and Verification of Dynamic Communication Systems. In Kees Goossens and Laure Petrucci, editors, *Proc. ACSD 2006*, Turku, Finland, June 2006. IEEE.

3. D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *Journal of the Association for Computing Machinery*, 30(2):323–342, April 1983.
4. E. M. Clarke, M. Talupur, and H. Veith. Environment Abstraction for Parameterized Verification. In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI 2006, USA, January 2006*, volume 3855 of *LNCS*, pages 126–141. Springer, 2006.
5. Werner Damm and Bernd Westphal. Live and let die: LSC-based verification of UML-models. *Science of Computer Programming*, 55(1–3):117–159, March 2005.
6. M. Frodigh, P. Johansson, and P. Larsson. Wireless Ad Hoc Networking: The Art of Networking without a Network. *Ericsson Review*, 4, 2000.
7. V. Gyuris and A. P. Sistla. On-the-Fly Model Checking Under Fairness that Exploits Symmetry. *Formal Methods in System Design*, 15(3):217–238, Nov 1999.
8. Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2004. HOL g 03:1 1.Ex.
9. Ann Hsu, Farokh Eskafi, Sonia Sachs, and Pravin Varaiya. The Design of Platoon Maneuver Protocols for IVHS. PATH Research Report UCB-ITS-PRR-91-6, Inst. of Transportation Studies, University of California, April 1991. ISSN 1055-1425.
10. Y. Kesten, A. Pnueli, E. Shahar, and L. D. Zuck. Network Invariants in Action. In *Proc. CONCUR'02*, pages 101–115, London, UK, 2002. Springer-Verlag.
11. Boris D. Lubachevsky. An Approach to Automating the Verification of Compact Parallel Coordination Programs I. *Acta Inf.*, 21:125–169, 1984.
12. Ken L. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37:279–309, 2000.
13. R. Milner. *Communicating and Mobile Systems: The Pi Calculus*. CU Press, 1999.
14. Amir Pnueli, Jessie Xu, and Lenore Zuck. Liveness with  $(0,1,\infty)$ -Counter Abstraction. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV 2003, Boulder, CO, USA, July 2003*, volume 2725 of *LNCS*, pages 107–133. Springer, 2003.
15. Jan Rakow. Verification of Dynamic Communication Systems. Master's thesis, Carl von Ossietzky Universität Oldenburg, April 2006.
16. S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In O. Grumberg, editor, *Proc. CAV'97*, volume 1254, pages 72–83. Springer, 1997.
17. S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric Shape Analysis via 3-Valued Logic. *ACM Transactions on Programming Languages and Systems*, 22, 2001.
18. UNISIG. SUBSET 026-Chapter 3; Version 2.2.2 (SRS), March 2002.  
<http://www.aeif.org/ccm/default.asp>.
19. Björn Wachter and Bernd Westphal. The Spotlight Principle. On Combining Process-Summarising State Abstractions. In Byron Cook and Andreas Podelski, editors, *VMCAI'07, France*, volume 4349 of *LNCS*, pages 182–198. Springer, 2007.
20. Bernd Westphal. LSC Verification for UML Models with Unbounded Creation and Destruction. In Willem Visser Byron Cook, Scott Stoller, editor, *Proc. SoftMC 2005*, volume 144(3) of *ENTCS*, pages 133–145. Elsevier B.V., July 2005.
21. Fei Xie and James C. Browne. Integrated State Space Reduction for Model Checking Executable Object-oriented Software System Designs. In Ralf-Detlef Kutsche and Herbert Weber, editors, *FASE 2002, Grenoble, France, April 8-12*, number 2306 in *LNCS*, pages 64–79. Springer-Verlag, 2002.
22. E. Yahav and G. Ramalingam. Verifying safety properties using separation and heterogeneous abstractions. In *Proc. of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, pages 25–34. ACM Press, 2004.
23. Eran Yahav. Verifying safety properties of concurrent Java programs using 3-valued logic. *ACM SIGPLAN Notices*, 36(3):27–40, 2001.