Counterexample Guided Spotlight Abstraction Refinement*

Tobe Toben

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

Abstract. This paper addresses the formal verification of distributed systems comprising a dynamically changing and potentially unbounded number of processes. We employ the spotlight principle to obtain a concise finitary abstraction of the system and devise an abstraction refinement strategy guided by the analysis of abstract counterexamples.

It turns out that the key problem for spotlight refinement is the identification of spurious counterexamples. We observe that the problem is in general undecidable, and provide a sound but incomplete method that is able to solve the problem for many practically relevant systems. Our method is driven by a *three-valued* satisfaction relation for temporal specifications that accounts for the fact that concrete counterexamples can be identified in the abstracted system if they occur within the spotlight.

1 Introduction

Distributed systems comprising a dynamically changing and potentially unbounded number of processes naturally occur in various areas of ubiquitous computing, ad-hoc networking and traffic management systems. For example, processes may represent mobile devices entering a wireless network, or trains approaching a railway controller that is responsible for granting movement authorisations (as proposed e.g. in the ETCS Level 3 standard [1]). The correct treatment of at run-time appearing and disappearing processes adds a new level of complexity when designing safety-critical distributed systems.

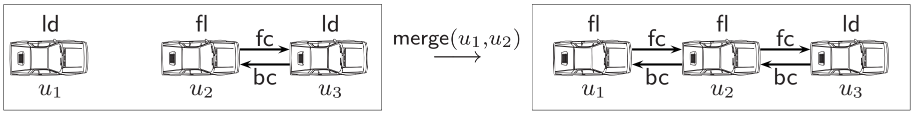
The use of formal methods can help to avoid errors early in the system development phase. Formal verification of *dynamic* behaviour however imposes two challenges. Firstly, it requires an appropriate formal description of the system behaviour. This formalism has to go beyond standard notations for reactive systems like Kripke structures [2,3], because the local states of arbitrary many alive processes have to be representable. Secondly, automatic verification techniques like model-checking [3] are a priori only applicable to (small) finite-state systems. One approach to deal with this problem is to use finitary abstraction [4], that is, to devise a finite abstraction of the system and to show that the analysis of the abstract system is sufficient to ensure the correctness of the original system.

* 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).

This paper proposes a solution for both of these problems. Inspired by early work in the area of first-order modal logic [5], we use *first-order logical structures* as a formal representation of a global system state. These structures comprise a set of process identities and an interpretation of predicates for these process identities. With this, the behaviour of a dynamic system can be represented as an infinite-state transition system over logical structures. Consequently, we use a *first-order* variant of linear temporal logic for the formal requirement specification, that is, we allow to quantify over variables denoting process identities.

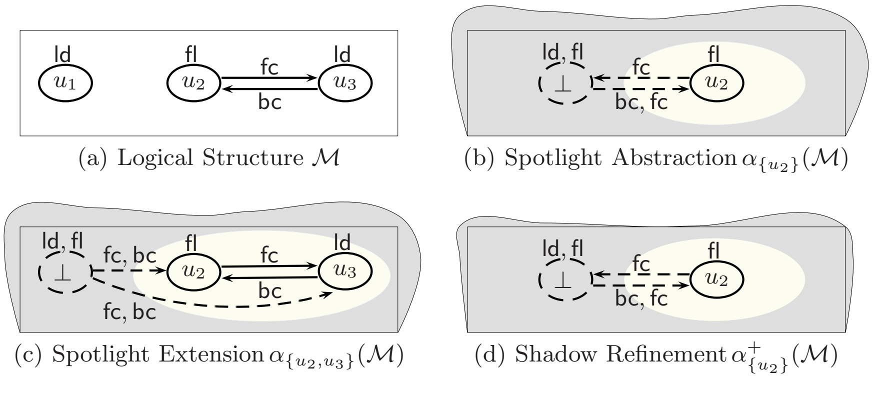
By a finitary abstraction of the considered systems, we are able to use any of the highly optimised verification engines (like VIS [6] or SPIN [7]) that are available for finite-state systems. The employed abstraction follows the spotlight principle [8] by representing only a finite number of processes exact and collapsing the rest into one dedicated *summary process.* The number of concrete processes can easily be determined by the number of variables in the requirement specification. Formally, the abstraction yields *three-valued* logical structures, because the predicate interpretation for the summary process may neither become *true* nor *false* but "*maybe*" in order to remain sound. The abstract system yields a sound but incomplete overapproximation of the original system, i.e. the satisfaction of properties transfers from the abstract to the original system, but in general not vice versa: Not every property that is valid for the original system can be proven in the abstraction. This entails the existence of *spurious counterexamples* which demonstrate the violation of a property in the abstraction, although the property actually holds for the original system. Thus, an abstract counterexample can not be "trusted" unless it has been validated. However, due to the heterogeneous nature of the underlying abstraction, we are also able to obtain *concrete* counterexamples directly in the abstracted system, namely if they occur within the spotlight part of the abstraction. We will formalise this intuition in the course of this paper, again by the usage of three-valued logic.

Running Example. We use the car platooning scenario to illustrate our approach. In this case study, cars driving on a highway are supposed to autonomously form car platoons, i.e. series of interlinked cars driving with only little distance. To do so, a car can *merge* with a car driving in front (cf. Fig. 1), and a car being the head of a platoon can *split* from its followers. As cars can freely enter and leave a highway, no finite upper bound on the number of cars can be made.



(a) Car u_1 approaching the platoon. (b) Car u_1 has merged with u_2.

Fig. 1. Car platooning. A car at the head of a platoon is called a leader (ld), where a single car is represented as a platoon of size one. A car driving within a platoon is called a follower (fl). The platoon itself is organised as a doubly-linked list, where each car has a (communication) link to its front car (fc) and a link to its back car (bc).



(a) Logical Structure $\mathcal{M}$ (b) Spotlight Abstraction $\alpha_{\{u_2\}}(\mathcal{M})$

(c) Spotlight Extension $\alpha_{\{u_2,u_3\}}(\mathcal{M})$ (d) Shadow Refinement $\alpha^{+}_{\{u_2\}}(\mathcal{M})$

Fig. 2. Spotlight abstraction (cf. Def. 2) and possible refinements

Spotlight Abstraction and Refinement. Figure 2(a) graphically represents the logical structure according to the global state of the car platooning system in Fig. 1(a). It comprises three process identities, u_1, u_2 and u_3, and provides the predicate interpretation by labelling the nodes and arcs by those predicates that are true for these nodes. For example, the unary predicate ld is true for u_1 and u_3 and false for u_2, and the binary predicate fc is true only for (u_2, u_3).

Figure 2(b) shows the abstraction of this structure with only u_2 in the spotlight. The truth values of the predicates are kept only among the processes in the spotlight, i.e. the abstraction preserves that there exists a *follower* car with a fc link to (at least) one abstract car. Any information about the processes in the shadows however is neglected, i.e. any predicate yields *maybe* for the summary process $\perp$, as indicated by dashed lines. Hence the summary process in general considerably *overapproximates* the original structure, as indicated by the gray area exceeding the box. This coarse representation is special to spotlight abstraction and is the key for easily obtaining the abstract transition system. It provides a sound abstraction, thus every temporal specification that holds for the spotlight abstracted transition system also holds for the original transition system. Besides overapproximating the shadows, the spotlight abstraction also maintains an *underapproximation* of the original system regarding the finite set of spotlight processes. Thus, a natural refinement of the abstraction consists of enlarging the spotlight. In Fig. 2(c), the spotlight comprises u_2 and u_3, and the abstraction now preserves the existence of a valid platoon of size two. But note that we can only ensure the validity of this spotlight configuration if the system run leading to Fig. 2(c) is not illegally influenced by the summary process.

In general, the overapproximative behaviour of $\perp$ may result in spotlight configurations that are not reachable in the original system. Another important refinement is thus to eliminate spurious behaviour of the summary process, as graphically indicated by a reduced gray area in Fig. 2(d). We will use temporal assumptions for refining the behaviour of $\perp$, stating that certain interactions of $\perp$ are not possible in certain spotlight configurations. These assumptions will be derived from the validation of abstract counterexamples that have been obtained for the verification of a given requirement specification. Following the discussion above, we propose to regard the satisfaction of a temporal specification in a three-valued fashion, namely as *true* if it is satisfied in all system runs, *false* if it is violated in some run independent of $\perp$, and *unknown* else. In the latter case, we aim at concretising the abstract counterexample via spotlight extension, that is, we try to reproduce the behaviour of $\perp$ by concrete processes. For this, we add additional spotlight processes. If the behaviour is not possible with concrete processes, an assumption for shadow refinement has been obtained and a new verification task is started under the refined abstraction. If the behaviour can be validated with concrete processes, we may obtain a concrete counterexample and are done. But it is also possible that the behaviour of the new spotlight processes is again influenced by the abstract process, hence we may obtain another abstract counterexample that itself has to be validated. Therefore, we *successively* add concrete processes until a definite answer has been obtained.

The major contribution of this paper is an instantiation of the framework of counterexample guided abstraction refinement [9] for spotlight abstraction, that is, we automate the refinement of spotlight abstraction. We observe that, due to the coarse abstraction, the validation of abstract counterexamples becomes difficult (and undecidable in general) while the shadow refinement can be shown to be very effective. We devise a translation from counterexamples to temporal specifications that on the one hand allows us to validate the counterexample, and on the other hand is a source for a refinement assumption. The translation and refinement loop has been evaluated on the basis of a verification toolset [10] for dynamic systems, and first experimental results are given in Sect. 5.

Related Work. In [11], spotlight abstraction is applied for the verification of UML models, and the abstraction is manually refined by separately established assumptions. Our approach allows us to compute such kind of assumptions automatically. [12] uses a variant of spotlight abstraction for the verification of parameterised communication models, but they leave out abstraction refinement as future work. [13] proposes a general strategy for spotlight abstraction refinement by inferring and integrating so-called non-interference lemmata. This idea is realised in [14] resp. [15], where two particular kinds of invariants, namely non-interference properties resp. topology invariants, are automatically computed and integrated into a refinement procedure. These approaches however have no immediate potential for iteration, i.e. if the refinement by the inferred invariants is not accurate enough to prove the specification, one remains inconclusive.

Other approaches for analysis of dynamic systems work on graph transformation systems and define tailored abstraction techniques like Partner Abstraction [16] or approximation in terms of Petri nets [17]. The latter approach also applies a CEGAR loop to reduce spurious behaviour stemming from the merge of graph nodes during the abstraction. However, the unique nature of the spotlight abstraction principle requires new validation and refinement strategies.

Abstracting a set of concrete nodes to summary nodes is also the underlying principle of parametric shape analysis [18]. Their abstraction mechanism is more precise (and therefore more expensive) by creating *multiple* summary nodes for different equivalence classes of concrete nodes. However, concrete nodes may migrate from one summary process to another, thereby losing their identity (cf. [8]). In contrast, spotlight abstraction allows us to trace process identities over the time, which enables the analysis of full temporal properties. On the other hand, our logic is not expressive enough to reason about the shape of the *overall* heap structure, because the transitive closure operator does not fit well with spotlight abstraction.

2 Preliminaries

In general, abstraction comes hand in hand with a loss of information. To formally characterise partial impreciseness, we use the framework of three-valued logic according to Kleene [19]. Here, the boolean domain comprises three values, namely $\mathbb{B}_3 := \{0, 1/2, 1\}$. Besides the *value order* $\leq$ on $\mathbb{B}_3 \times \mathbb{B}_3$, we consider the *information order* $\sqsubseteq$ on $\mathbb{B}_3 \times \mathbb{B}_3$ defined as $b_1 \sqsubseteq b_2$ iff $b_1 = 1/2$ or $b_1 = b_2$.

As we do not impose an upper bound on the number of currently alive processes, we assume an infinite set $Id = \{u_1, u_2, \ldots\}$ of *process identities*. By $\perp \notin Id$ we denote the *summary process*, and we set $I^\perp := I \dot{\cup} \{\perp\}$ for any $I \subseteq Id$. The actual configuration and evolution of the system will be characterised by a number of predicates, i.e. we define a *signature* $\mathcal{S} = (\mathcal{X}, \mathcal{P}_S, \mathcal{P}_L, \mathcal{P}_E)$ as a collection of a finite set of logical variables $\mathcal{X}$, a finite set of unary state predicates $\mathcal{P}_S$, a finite set of binary link predicates $\mathcal{P}_L$, and a finite set of evolution predicates $\mathcal{P}_E$. For convenience, we set $\mathcal{P}_{SL} := \mathcal{P}_S \cup \mathcal{P}_L$, and $\mathcal{P} := \mathcal{P}_S \cup \mathcal{P}_L \cup \mathcal{P}_E$, and denote the arity of a predicate $p \in \mathcal{P}$ by k_p. With this, a configuration of the system can be faithfully represented as a first-order logical structure, i.e. a tuple $(\mathcal{U}, \iota)$ comprising a set of currently alive processes $\mathcal{U} \subseteq Id^\perp$ and an interpretation of the state and link predicates, i.e. ι yields for each $p \in \mathcal{P}_{SL}$ a function $\iota(p) : \mathcal{U}^{k_p} \rightharpoonup \mathbb{B}_3$. For a subset of identities $I \subseteq Id^\perp$, we use $\mathbf{M}_\mathcal{S}(I) := \{(\mathcal{U}, \iota) \mid \mathcal{U} \subseteq I\}$ to denote the set of logical structures where at most the identities from I are present. In the following, we will represent an interpretation ι by the tuple $(\iota_\mathbf{1}, \iota_{\mathbf{1/2}})$ with

$$\begin{aligned}\iota_\mathbf{1} &:= \{p(u_1, \ldots, u_{k_p}) \mid \iota(p)(u_1, \ldots, u_{k_p}) = 1\}\\ \iota_{\mathbf{1/2}} &:= \{p(u_1, \ldots, u_{k_p}) \mid \iota(p)(u_1, \ldots, u_{k_p}) = 1/2\}.\end{aligned}$$

3 Dynamic Systems

The behaviour of a dynamic system can be formally characterised by a (infinitely large) labelled transition system where the states are logical structures and the transitions are labelled by evolution predicates. To actually model such systems, we introduce a symbolic description of a dynamic system as a set of evolution rules D, each of them comprising a *label*, a *guard* and a sequence of *actions*. The label is a *term* over evolution predicates $\mathcal{P}_E$, i.e. it is of the form $p(x_1, \ldots, x_{k_p})$ with $x_i \in \mathcal{X}$. The guard is a *formula* over state and link predicates, generated by the grammar $\psi ::= t \mid x_1 = x_2 \mid \neg\psi \mid \psi_1 \wedge \psi_2$ where t is a term over $\mathcal{P}_{SL}$. Finally, an action sequence is generated by the grammar $a ::= a_1; a_2 \mid t \mid \neg t \mid \circledast x \mid \ominus x$. Here, a positive term t over $\mathcal{P}_{SL}$ turns the corresponding predicate to true, and a negative term $\neg t$ sets it to false. The action $\circledast x$ will create a new process denoted by $x \in \mathcal{X}$, and $\ominus x$ will kill the corresponding process. We require that each label comprises exactly those variables that are used in its guard and in its actions.

Before defining the formal semantics, let us formalise the car platooning example as a dynamic system Car over signature $(\mathcal{X}, \mathcal{P}_S, \mathcal{P}_L, \mathcal{P}_E)$ with $\mathcal{P}_S = \{\mathsf{ld}, \mathsf{fl}\}$, $\mathcal{P}_L = \{\mathsf{bc}, \mathsf{fc}\}$ and $\mathcal{P}_E = \{\mathsf{new}/1, \mathsf{merge}/2, \mathsf{split}/2\}$. The evolution rules are

$$\begin{aligned}
\mathsf{new}(x) \bullet \neg\mathsf{alive}(x) &\blacktriangleright \circledast x; \mathsf{ld}(x) \\
\mathsf{merge}(x_1, x_2) \bullet \mathsf{ld}(x_1) \wedge \mathsf{alive}(x_2) \wedge x_1 \neq x_2 &\blacktriangleright \neg\mathsf{ld}(x_1); \mathsf{fl}(x_1); \mathsf{fc}(x_1, x_2); \mathsf{bc}(x_2, x_1) \\
\mathsf{split}(x_1, x_2) \bullet \mathsf{ld}(x_1) \wedge \mathsf{bc}(x_1, x_2) &\blacktriangleright \mathsf{ld}(x_2); \neg\mathsf{fl}(x_2); \neg\mathsf{fc}(x_2, x_1); \neg\mathsf{bc}(x_1, x_2)
\end{aligned}$$

written in the form *label* $\bullet$ *guard* $\blacktriangleright$ *actions*. The first rule allows to freely create cars as leaders, that is, any structure where some process identity u is currently not alive (see below for the definition of alive) may evolve into a structure where u exists and $\mathsf{ld}(u)$ holds. The second rule allows to merge a leader car with some other alive car, that is, the leader becomes a follower and communication links are established. The third rule allows a leader car to split from its back car.

Now to formally characterise when two logical structures are in transition relation according to a dynamic system, we need to define the satisfaction of a guard and the effect of applying actions to a logical structure. Let $\mathcal{M} = (\mathcal{U}, \iota) \in \mathbf{M}_S(I)$ be a logical structure and $\mathcal{V} \in \mathit{Vals}_I(X)$ a valuation, i.e. a function $X \to I$ of variables $X \subseteq \mathcal{X}$ to identities $I \subseteq \mathit{Id}^{\perp}$. Then

$$\begin{aligned}
\mathcal{M}[\![p(x_1, \ldots, x_{k_p})]\!](\mathcal{V}) &:= \mathcal{V}(x_1), \ldots, \mathcal{V}(x_{k_p}) \in \mathcal{U} \wedge \iota(p)(\mathcal{V}(x_1), \ldots, \mathcal{V}(x_{k_p})) \\
\mathcal{M}[\![x_1 = x_2]\!](\mathcal{V}) &:= \mathcal{V}(x_1), \mathcal{V}(x_2) \in \mathcal{U} \wedge \mathcal{V}(x_1) = \mathcal{V}(x_2) \\
\mathcal{M}[\![\neg\psi]\!](\mathcal{V}) &:= \neg\mathcal{M}[\![\psi]\!](\mathcal{V}) \\
\mathcal{M}[\![\psi_1 \wedge \psi_2]\!](\mathcal{V}) &:= \mathcal{M}[\![\psi_1]\!](\mathcal{V}) \wedge \mathcal{M}[\![\psi_2]\!](\mathcal{V})
\end{aligned}$$

inductively defines the (possibly three-valued) *satisfaction* of a guard. We decided that a term can only be satisfied if all its arguments are currently alive, that is, in $\mathcal{U}$. In particular, this allows for the abbreviation $\mathsf{alive}(x) := x{=}x$. The action *update* of $\mathcal{M} = (\mathcal{U}, \iota)$ under valuation $\mathcal{V}$ is inductively defined as

$$\begin{aligned}
\mathcal{M}\langle a_1; a_2; \ldots; a_n\rangle(\mathcal{V}) &:= \mathcal{M}\langle a_1\rangle(\mathcal{V})\langle a_2; \ldots; a_n\rangle(\mathcal{V}) \\
\mathcal{M}\langle p(x_1, \ldots, x_{k_p})\rangle(\mathcal{V}) &:= (\mathcal{U}, \iota[p \mapsto \iota(p)[(\mathcal{V}(x_1), \ldots, \mathcal{V}(x_{k_p})) \mapsto 1]]) \\
\mathcal{M}\langle \neg p(x_1, \ldots, x_{k_p})\rangle(\mathcal{V}) &:= (\mathcal{U}, \iota[p \mapsto \iota(p)[(\mathcal{V}(x_1), \ldots, \mathcal{V}(x_{k_p})) \mapsto 0]]) \\
\mathcal{M}\langle \circledast x\rangle(\mathcal{V}) &:= (\mathcal{U} \cup \{\mathcal{V}(x)\}, \iota) \\
\mathcal{M}\langle \ominus x\rangle(\mathcal{V}) &:= (\mathcal{U} \setminus \{\mathcal{V}(x)\}, \iota)
\end{aligned}$$

where $f[x \mapsto y]$ denotes substitution for some function $f : X \to Y$, i.e. it alters the function f to yield $y \in Y$ for argument $x \in X$, and $f(x')$ for all $x' \in X \setminus \{x\}$.

Starting at the empty structure $(\emptyset, (\emptyset, \emptyset))$, the semantics of a dynamic system is computed by iteratively applying evolution rules that are *enabled*, i.e. there is a valuation $\mathcal{V}$ into the infinite domain of identities *Id* s.t. the guard is satisfied. Note that the evolution steps are labelled by the applied rule and the subset of involved identities. This information will be exploited later when evaluating temporal specifications and for performing spotlight extension as refinement.

Definition 1 (Concrete Semantics). *The concrete semantics of a dynamic system* D *over* $\mathcal{S}$*, denoted* $[\mathsf{D}]$*, is the labelled transition system* $(\mathbf{S}, \mathbf{S}_0, \mathbf{L}, \mathbf{R})$ *with*

– *states* $\mathbf{S} := \mathbf{M}_{\mathcal{S}}(Id)$ *with initial state* $\mathbf{S}_0 := (\emptyset, (\emptyset, \emptyset))$,
– *labels* $\mathbf{L}$ *and transitions* $\mathbf{R} := \{(\mathcal{M}, \mathit{label}[V], \mathcal{M}\langle \mathit{actions}\rangle(\mathcal{V})) \in \mathbf{S} \times \mathbf{L} \times \mathbf{S} \mid \exists\,(\mathit{label} \bullet \mathit{guard} \blacktriangleright \mathit{actions}) \in \mathsf{D}, \mathcal{V} \in \mathit{Vals}_{Id}(\mathcal{X}) : \mathcal{M}[\![\mathit{guard}]\!](\mathcal{V})\}.$

where $p(x_1, \ldots, x_{k_p})[\mathcal{V}] := p(\mathcal{V}(x_1), \ldots, \mathcal{V}(x_{k_p}))$ *for* $p \in \mathcal{P}_E$. ♢

The concrete semantics induces a set of runs of a dynamic system D as follows. A run of $\mathsf{T} = (\mathbf{S}, \mathbf{S}_0, \mathbf{L}, \mathbf{R})$ is an infinite sequence $((L_i, S_i))_{i \in \mathbb{N}_0}$ of labels $L_i \in \mathbf{L}$ and states $S_i = (\mathcal{U}_i, \iota_i) \in \mathbf{S}$ such that $S_0 = \mathbf{S}_0$ and $(S_i, L_{i+1}, S_{i+1}) \in \mathbf{R}$ for all $i \geq 0$. The runs of T are denoted by *Runs*(T). An example run of $[\mathsf{Car}]$ is

$$\begin{aligned}
&((\emptyset, (\emptyset, \emptyset))),\\
(\mathsf{new}(u_1),\ &(\{u_1\}, (\{\mathsf{ld}(u_1)\}, \emptyset))),\\
(\mathsf{new}(u_2),\ &(\{u_1, u_2\}, (\{\mathsf{ld}(u_1), \mathsf{ld}(u_2)\}, \emptyset))),\\
(\mathsf{merge}(u_1, u_2),\ &(\{u_1, u_2\}, (\{\mathsf{fl}(u_1), \mathsf{ld}(u_2), \mathsf{fc}(u_1, u_2), \mathsf{bc}(u_2, u_1)\}, \emptyset))),\\
(\mathsf{split}(u_2, u_1),\ &(\{u_1, u_2\}, (\{\mathsf{ld}(u_1), \mathsf{ld}(u_2)\}, \emptyset))),\ \ldots
\end{aligned}$$

where two cars u_1, u_2 appear, merge to a platoon of size two and split again.

Note that the unbounded number of processes in a dynamic system renders the verification problem undecidable. In [20] we show how to encode the transitions of a two-counter-machine by a set of evolution rules as introduced above. The basic idea is to simulate an unbounded counter as a linked list of processes.

3.1 Spotlight Abstraction of Dynamic Systems

To obtain a finite representation of the infinite-state transition system, we apply spotlight abstraction [8]. It takes a finite set of "spotlight identities" $I \subseteq Id$ and collapses all identities from $Id \setminus I$ into the abstract identity $\perp$, for which all predicates then evaluate to $1/2$ in order to obtain a sound abstraction.

Definition 2 (Spotlight Abstraction). *The spotlight abstraction of a logical structure is the function* $\alpha.(\cdot) : 2^{Id} \times \mathbf{M}_{\mathcal{S}}(Id^{\perp}) \to \mathbf{M}_{\mathcal{S}}(Id^{\perp})$ *with* $\alpha_I((\mathcal{U}, \iota)) := (\alpha_I(\mathcal{U}), \alpha_I(\iota))$ *where* $\alpha_I(\mathcal{U}) := (\mathcal{U} \cap I) \cup \{\perp\}$ *and*

$$\begin{aligned}
\alpha_I(\iota)(p_l)(\perp, u_1) &:= 1/2 & \alpha_I(\iota)(p_s)(u_1) &:= \iota(p_s)(u_1)\\
\alpha_I(\iota)(p_l)(u_1, u_2) &:= \iota(p_l)(u_1, u_2) & \alpha_I(\iota)(p_s)(\perp) &:= 1/2\\
\alpha_I(\iota)(p_l)(u_1, \perp) &:= 1/2 \text{ if } \exists\, u' \in \mathcal{U} \setminus I : \iota(p_l)(u_1, u'), \text{ and } 0 \text{ else}
\end{aligned}$$

for $p_s \in \mathcal{P}_S$, $p_l \in \mathcal{P}_L$, *and* $u_1, u_2 \in \mathcal{U} \cap I$. ♢

Note that a binary predicate p_l for some concrete process u_1 and the summary process $\perp$ becomes $1/2$ if there was at least one collapsed process u' where the predicate was true. By this we actually lose the *number* of links to abstracted processes (cf. Fig. 2(b)). The fact that the abstraction neglects all information about the processes outside of the spotlight allows the abstract transition relation to be easily computed. We simply restrict the set of states to finite three-valued structures $\mathbf{M}_{\mathcal{S}}(I^{\perp})$, apply the action update as in the concrete semantics, and then blur the resulting structure via the abstraction function α_I from Def. 2.

Definition 3 (Abstract Semantics). *The abstract semantics of a dynamic system* $\mathsf{D} \in \mathcal{D}_{\mathcal{S}}$ *and a set of identities* $I \subseteq Id$*, denoted* $[\mathsf{D}]^{\sharp}_{I}$*, is the labelled transition system* $(\mathbf{S}, \mathbf{S}_0, \mathbf{L}, \mathbf{R})$ *with*

– *states* $\mathbf{S} := \mathbf{M}_{\mathcal{S}}(I^{\perp})$ *with initial state* $\mathbf{S}_0 := \alpha_I((\emptyset, (\emptyset, \emptyset)),$
– *and transitions* $\mathbf{R} := \{(\mathcal{M}, \mathit{label}[V], \alpha_I(\mathcal{M}\langle \mathit{actions}\rangle(\mathcal{V}))) \in \mathbf{S} \times \mathbf{L} \times \mathbf{S} \mid$
$\exists\, (\mathit{label} \bullet \mathit{guard} \blacktriangleright \mathit{actions}) \in \mathsf{D}, \mathcal{V} \in \mathit{Vals}_{I^{\perp}}(\mathcal{X}) : \mathcal{M}[\![\mathit{guard}]\!](\mathcal{V}) \geq 1/2\}.$ $\Diamond$

Remark 1. Let D be a dynamic system and $I \subset Id$ a finite set of identities, i.e. a finite spotlight. Then $[\mathsf{D}]^{\sharp}_{I}$ is finite, i.e. it comprises only finitely many states. $\Diamond$

We proceed by introducing the syntax and semantics of a specification logic for dynamic systems, and provide a generalised soundness theorem for spotlight abstraction in terms of the information order of three-valued logic.

3.2 Specification Logic for Dynamic Systems

Temporal logic [21] has become a standard formalism to reason about system behaviour. In this paper, we use a variant of first-order linear time logic with implicit universal quantification. The specification language over signature $\mathcal{S} = (\mathcal{X}, \mathcal{P}_S, \mathcal{P}_L, \mathcal{P}_E)$, denoted $\mathit{Specs}_{\mathcal{S}}$, is defined by the grammar

$$\phi ::= \mathbf{tt} \mid t \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi \geq 1/2 \mid \mathsf{G}\,\phi \mid \mathsf{F}\,\phi$$

where t is a term over $\mathcal{P}$. For example, the following specification for the Car system states that for all cars x_1, whenever x_1 is in its ld ("leader") state it has no fc ("front car") connection to any car x_2:

$$\phi_{\mathsf{ld}} := \mathsf{G}\,(\mathsf{ld}(x_1) \rightarrow \neg\mathsf{fc}(x_1, x_2))$$

In the case of overapproximative abstraction, the satisfaction of a temporal specification transfers from the abstract to the concrete system, i.e. $\mathsf{D}^{\sharp} \models \phi \rightarrow \mathsf{D} \models \phi$. In the case of spotlight abstraction, we observe special cases where also the converse holds, i.e. where $\mathsf{D}^{\sharp} \not\models \phi \rightarrow \mathsf{D} \not\models \phi$. We can easily identify these cases by exploiting the fact that the transitions are annotated by the set of involved identities: Any violation where $\perp$ is not involved is by construction a concrete violation in the original system. This finding leads to a new three-valued definition of a run satisfying a temporal specification as follows, and is the basis for identifying non-spurious abstract counterexamples.

Definition 4 (Satisfaction Relation). *Let* T *be a labelled transition system over signature* $\mathcal{S}$. *The satisfaction of* $\phi \in \mathit{Specs}_{\mathcal{S}}$ *in a run* $\pi = ((L_i, S_i))_{i \in \mathbb{N}_0} \in \mathit{Runs}(\mathsf{T})$ *under valuation* $\mathcal{V} \in \mathit{Vals}_{\mathit{Id}^{\perp}}(\mathit{vars}(\phi))$ *is defined inductively as follows, where* $p_s \in \mathcal{P}_S$, $p_l \in \mathcal{P}_L$ *and* $p_e \in \mathcal{P}_E$.

$$\pi[\![p_s(x)]\!]^i(\mathcal{V}) := S_i[\![p_s(x)]\!](\mathcal{V}) \qquad \pi[\![\phi \geq 1/2]\!]^i(\mathcal{V}) := \pi[\![\phi]\!]^i(\mathcal{V}) \geq 1/2$$

$$\pi[\![p_l(x_1, x_2)]\!]^i(\mathcal{V}) := S_i[\![p_l(x_1, x_2)]\!](\mathcal{V}) \qquad \pi[\![\neg\phi]\!]^i(\mathcal{V}) := \neg\, \pi[\![\phi]\!]^i(\mathcal{V})$$

$$\pi[\![p_e(x_1, \ldots, x_{k_{p_e}})]\!]^i(\mathcal{V}) := L_i = p_e(\mathcal{V}(x_1), \ldots, \mathcal{V}(u_{k_{p_e}})) \qquad \pi[\![\mathtt{tt}]\!]^i(\mathcal{V}) := 1$$

$$\pi[\![\phi_1 \wedge \phi_2]\!]^i(\mathcal{V}) := \pi[\![\phi_1]\!]^i(\mathcal{V}) \wedge \pi[\![\phi_2]\!]^i(\mathcal{V})$$

$$\pi[\![\mathsf{G}\,\phi]\!]^i(\mathcal{V}) := \begin{cases} 1 & \text{if } \forall\, k \geq i : \big(\pi[\![\phi]\!]^k(\mathcal{V}) = 1\big) \\ 0 & \text{if } \exists\, k \geq i : \big(\pi[\![\phi]\!]^k(\mathcal{V}) = 0 \,\wedge \\ & \qquad \forall\, j \in \{i \ldots k\} : \perp \notin L_j\,\big) \\ 1/2 & \text{else} \end{cases}$$

$$\pi[\![\mathsf{F}\,\phi]\!]^i(\mathcal{V}) := \neg\, \pi[\![\mathsf{G}\,\neg\phi]\!]^i(\mathcal{V})$$

The satisfaction of ϕ *in* T *under* $\mathcal{V}$ *is defined as the minimum over all runs:*

$$\mathsf{T}[\![\phi]\!](\mathcal{V}) := \min\{\pi[\![\phi]\!]^0(\mathcal{V}) \in \mathbb{B}_3 \mid \pi \in \mathit{Runs}(\mathsf{T})\}. \qquad \Diamond$$

By this, we obtain an *embedding* (with respect to the information order) of the three-valued satisfaction for the abstracted semantics into the satisfaction for the concrete semantics. That is, whenever $[\mathsf{D}]^{\sharp}_{I}[\![\phi]\!](\mathcal{V})$ evaluates to a definite value, then $[\mathsf{D}][\![\phi]\!](\mathcal{V})$ evaluates to the same value. If $[\mathsf{D}]^{\sharp}_{I}[\![\phi]\!](\mathcal{V}) = 1/2$, we remain inconclusive. The following theorem formally states this property.

Theorem 1 (Embedding). *Let* D *be a dynamic system over* $\mathcal{S}$ *and* $\phi \in \mathit{Specs}_{\mathcal{S}}$. *Then*

$$[\mathsf{D}]^{\sharp}_{I}[\![\phi]\!](\mathcal{V}) \;\sqsubseteq\; [\mathsf{D}][\![\phi]\!](\mathcal{V})$$

for any spotlight $I \subseteq \mathit{Id}$ *and valuation* $\mathcal{V} \in \mathit{Vals}_I(\mathit{vars}(\phi))$. $\Diamond$

Based on the satisfaction relation from Def. 4, we define the satisfaction of the quantified specification for a dynamic system in two variants, namely for the concrete and the abstract semantics of a dynamic system. For the latter, we allow to fix a subset of the variables that will bind to the $\perp$ identity, and we set the range of the actual valuation function as the content of the spotlight.

Definition 5 (Quantified Satisfaction). *For a dynamic system* D *over signature* $\mathcal{S}$, *a specification* $\phi \in \mathit{Specs}_{\mathcal{S}}$ *and variables* $X \subseteq \mathit{vars}(\phi)$ *we define*

$$\mathsf{D}[\![\phi]\!] := \min\{\, [\mathsf{D}][\![\phi]\!](\mathcal{V}) \in \mathbb{B}_3 \mid \mathcal{V} \in \mathit{Vals}_{\mathit{Id}}(\mathit{vars}(\phi))\}$$

$$\mathsf{D}^{\sharp}_{X}[\![\phi]\!] := \min\{\, [\mathsf{D}]^{\sharp}_{\mathsf{ran}(\mathcal{V})\setminus\{\perp\}}[\![\phi]\!](\mathcal{V}) \in \mathbb{B}_3 \mid \mathcal{V} \in \mathit{Vals}_{\mathit{Id}^{\perp}}(\mathit{vars}(\phi)) \text{ with } \mathcal{V}(x) = \perp \iff x \in X\}. \qquad \Diamond$$

Note that $\mathsf{D}[\![\phi]\!] \in \{0,1\}$. For the abstract semantics, we compute the minimal value according to $\leq$, i.e. if there is an abstract run and a valuation that yields a definite violation, we obtain a definite violation of the quantified specification, and if all runs under all valuations yield a definite satisfaction, we obtain a definite satisfaction. In all other cases, we obtain the indefinite value $1/2$.

The above definition requires to analyse the system under *infinitely many* valuation functions. However, we observe that dynamic systems induce transition systems that are symmetric in identities [22], i.e. whenever a set of processes I satisfies (violates) a specification, then any permutation on process identities $\sigma(I)$ satisfies (violates) the specification. This is because the behaviour of a process does not depend on its actual identity. Given a specification comprising N variables, we may reduce the number of valuations to a finite number N' of representative cases, where N' lies in $O(N!)$ [22]. These cases now only distinguish between the pairwise (in-)equality of process identities. For example, for the verification of ϕ_{ld} it is sufficient to consider two valuations, e.g. $[x_1 \mapsto u_1, x_2 \mapsto u_1]$ and $[x_1 \mapsto u_1, x_2 \mapsto u_2]$, because all other cases are symmetric. Note that in [23] the term Query Reduction was coined for such kind of exact reductions.

Theorem 1 directly transfers to the quantified case when no variable in $\mathit{vars}(\phi)$ evaluates to the $\perp$ identity, that is, $\mathsf{D}^\sharp_\emptyset[\![\phi]\!] \sqsubseteq \mathsf{D}[\![\phi]\!]$. In practice, this relation is only of interest when obtaining a definite value for the abstract system. However, the coarse representation of the spotlight environment allows for many (spurious) interferences with the spotlight (see an example below), hence we expect to often obtain the inconclusive result $1/2$. In the next section, we devise an iterative algorithm to suppress these interferences.

4 Spotlight Abstraction Refinement

In the following, let D be a dynamic system over signature $\mathcal{S}$ and $\phi \in \mathit{Specs}_{\mathcal{S}}$. Whenever $\mathsf{D}^\sharp_X[\![\phi]\!] \leq 1/2$, we can present a counterexample δ to demonstrate the (abstract) violation. By remark 1 such a counterexample can be finitely represented by a finite prefix of a run (possibly with a looping part as suffix, i.e. lasso-shaped [24]). We define the set of counterexamples

$$\delta = \langle \bar{\pi}, \mathcal{V} \rangle \in \mathit{Cex}(\mathsf{D}^\sharp_X[\![\phi]\!])$$

where $\bar{\pi} = ((L_i, S_i))_{0 \leq i \leq n}$ is a finite prefix of a run $\pi \in \mathit{Runs}([\mathsf{D}]^\sharp_I)$ and $\mathcal{V} \in \mathit{Vals}_{I^\perp}(\mathit{vars}(\phi))$ is a valuation such that $\pi[\![\phi]\!]^0(\mathcal{V}) \leq 1/2$.

A counterexample in $\mathit{Cex}(\mathsf{Car}^\sharp_\emptyset[\![\mathsf{G}(\mathsf{ld}(x_1) \rightarrow \neg\mathsf{fc}(x_1, x_2))]\!])$ is $\delta_{\mathsf{ld}} =$

$$\begin{aligned}
&\langle\ \alpha_{\{u_1,u_2\}}(\{\perp\}, (\emptyset, \emptyset)),\\
&(\mathsf{new}(u_1),\ \alpha_{\{u_1,u_2\}}(\{u_1, \perp\}, (\{\mathsf{ld}(u_1)\}, \emptyset))),\\
&(\mathsf{new}(u_2),\ \alpha_{\{u_1,u_2\}}(\{u_1, u_2, \perp\}, (\{\mathsf{ld}(u_1), \mathsf{ld}(u_2)\}, \emptyset))),\\
&(\mathsf{merge}(u_1, u_2),\ \alpha_{\{u_1,u_2\}}(\{u_1, u_2, \perp\}, \{\mathsf{fl}(u_1), \mathsf{ld}(u_2), \mathsf{fc}(u_1, u_2), \mathsf{bc}(u_2, u_1)\}, \emptyset)),\\
&(\mathsf{split}(\perp, u_1),\ \alpha_{\{u_1,u_2\}}(\{u_1, u_2, \perp\}, \{\mathsf{ld}(u_1), \mathsf{ld}(u_2), \mathsf{fc}(u_1, u_2), \mathsf{bc}(u_2, u_1)\}, \emptyset)),\\
&[x_1 \mapsto u_1, x_2 \mapsto u_2]\ \rangle
\end{aligned}$$

This run yields a possible violation of the specification because the last evolution $\mathsf{split}(\bot, u_1)$ yields a structure where $\mathsf{ld}(u_1) \wedge \mathsf{fc}(u_1, u_2)$, i.e. there is a leader car with a link to a front car. The question is whether those evolution transitions that affect the $\bot$ identity correspond to real behaviour of an (abstracted) process in the spotlight environment, or whether it is spurious behaviour stemming from the abstraction. In this example, we can manually argue that the $\mathsf{split}(\bot, u_1)$ is spurious because the prefix up to this transition indicates that no car in the spotlight environment is in a platoon with car u_1. In general, we consider a counterexample spurious if it has no concretisation, where a concretisation is possible if we can reproduce the behaviour of $\bot$ by concrete processes as follows.

Definition 6 (Concretisation). *Let* D *be a dynamic system over signature* $\mathcal{S}$. *A run* $\pi = ((L'_i, S'_i))_{i \in \mathbb{N}_0} \in \mathit{Runs}([\mathsf{D}])$ *is a concretisation of a counterexample*

$$\delta = \langle ((L_i, S_i))_{0 \le i \le n}, \mathcal{V} \rangle \in \mathit{Cex}(\mathsf{D}^\sharp_X [\![\phi]\!]),$$

written $\pi \succ \delta$, *if* $\pi[\![\phi]\!]^0(\mathcal{V}) = 0$ *and a monotone function* $f : \bot(\delta) \to \mathbb{N}$ *exists s.t.*

$$\forall\, i \in \mathsf{dom}(f) : L_i = L'_{f(i)}[\mathit{Id} \setminus \mathsf{ran}(\mathcal{V}) \mapsto \bot] \wedge S_i = \alpha_{\mathsf{ran}(\mathcal{V})}(S'_{f(i)})$$

where $\bot(\delta)$ *denotes the interferences of the abstract process in* δ, *i.e.* $\bot(\delta) := \{i \in \{1, \ldots, n\} \mid \bot \in L_i\}$. *The set of concretisations of* δ *is defined as* $\gamma(\delta) := \{\pi \in \mathit{Runs}([\mathsf{D}]) \mid \pi \succ \delta\}$, *and* δ *is called spurious, written* $\digamma(\delta)$, *if* $\gamma(\delta) = \emptyset$. $\diamondsuit$

The function f in Def. 6 ensures that each interference of the $\bot$ process is reproduced in the concretisation run in the same order. For example, Fig. 3 shows an abstract counterexample δ_{fl} for the specification $\phi_{\mathsf{fl}} := \mathsf{G}\,\neg\mathsf{fl}(x)$ where the only spotlight process u merges with the abstract process $\bot$. This counterexample is concretised by the run π_{fl} where the interference with the abstract process is replaced by interaction with a concrete process u'.

The identification of counterexample as being spurious requires to show that no concretisation exists, which reduces to an (in general undecidable) verification problem of the original system (see Lemma 1 below). However, the information contained in the counterexample allow a more specific verification task to be constructed where all evolution steps of the abstract process are now required to be performed by concrete processes. This provides a natural source for *spotlight extension* by introducing new variables in the specification as follows.

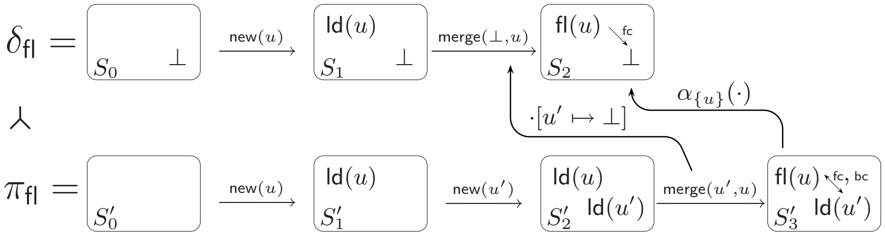


Fig. 3. Concretisation of an abstract counterexample for ϕ_{fl} with $f(2) = 3$

Definition 7 (Counterexample Formula). *Let* D *be a dynamic system over* $\mathcal{S}$ *and* $\delta = \langle((L_i, S_i))_{0 \le i \le n}, \mathcal{V}\rangle \in Cex(\mathsf{D}^\sharp_X[\![\phi]\!])$ *a counterexample. We define the* counterexample formula *of* δ *recursively as* $\varphi(\delta) := \varphi(\delta)^1$ *where*

$$\varphi(\delta)^i := \begin{cases} \mathsf{F}\left(label(L_i, \mathcal{V}, i) \wedge state(S_i, \mathcal{V}) \wedge (\varphi(\delta)^{i+1})\right) & \text{if } i \in \bot(\delta) \\ \varphi(\delta)^{i+1} & \text{if } i \notin \bot(\delta) \wedge i \le n \\ \mathbf{tt} & \text{else} \end{cases}$$

where

$$label(p_e(u_1, \ldots, u_{k_p}), \mathcal{V}, i) := p(\mathcal{V}_+^{-1}(u_1, i, 1), \ldots, \mathcal{V}_+^{-1}(u_{k_p}, i, k_p)), \text{ with}$$
$$\mathcal{V}_+^{-1}(u, i, j) := \begin{cases} \mathcal{V}^{-1}(u) & \text{if } u \neq \bot \\ x_{i,j} \in \mathcal{X} \setminus \mathsf{dom}(\mathcal{V}) & \text{else} \end{cases}$$

and

$$state(S, \mathcal{V}) := \bigwedge_{x_1 \in \mathsf{dom}(\mathcal{V}), p_s \in \mathcal{P}_S} \left(val(p_s(x_1), S, \mathcal{V}) \wedge \bigwedge_{x_2 \in \mathsf{dom}(\mathcal{V}), p_l \in \mathcal{P}_L} val(p_l(x_1, x_2), S, \mathcal{V})\right)$$

with $val(t, S, \mathcal{V}) = t$ *if* $S[\![t]\!](\mathcal{V}) = 1$ *and* $val(t, S, \mathcal{V}) = \neg t$ *else for any term* t*. By* $fresh(\varphi(\delta)) := vars(\varphi(\delta)) \setminus \mathsf{dom}(\mathcal{V})$ *we denote the new variables in* $\varphi(\delta)$*.* $\diamondsuit$

As an example, the counterexample δ_{ld} translates to $\varphi(\delta_{\mathsf{ld}}) =$

$$\mathsf{F}\ \big(\mathsf{split}(x_{4,1}, x_1) \wedge \mathsf{ld}(x_1) \wedge \neg\mathsf{fl}(x_1) \wedge \mathsf{fc}(x_1, x_2)) \wedge \neg\mathsf{bc}(x_1, x_2)) \wedge$$
$$\mathsf{ld}(x_2) \wedge \neg\mathsf{fl}(x_2) \wedge \neg\mathsf{fc}(x_2, x_1)) \wedge \mathsf{bc}(x_2, x_1) \wedge (\mathbf{tt})\big)$$

with $fresh(\varphi(\delta_{\mathsf{ld}})) = \{x_{4,1}\}$. The translation introduces nested F ("finally") expressions for each interference of the abstract $\bot$ process. In the *label* translation phase, each occurrence of $\bot$ in a transition label is substituted by a fresh process variable. The translation of the *state* part ensures that the configuration of the spotlight processes is preserved in the concretisation run. We have the following correspondence between the translation from Def. 7 and the definition of counterexample concretisation according to Def. 6.

Lemma 1 (Counterexample Validation). *Let* D *be a dynamic system over signature* $\mathcal{S}$*,* $\phi \in Specs_{\mathcal{S}}$ *and* $\delta \in Cex(\mathsf{D}^\sharp_X[\![\phi]\!])$ *a counterexample. Then*

$$\digamma(\delta) \iff \mathsf{D}[\![\neg\varphi(\delta) \vee \phi]\!]. \qquad \diamondsuit$$

In other words, if and only if no concrete run exists that both satisfies the counterexample formula and violates the specification, then the counterexample is spurious. For the car platooning case study, we can verify (cf. Sect. 5) that

$$\mathsf{Car}^\sharp_\emptyset[\![\neg\varphi(\delta_{\mathsf{ld}}) \vee \phi_{\mathsf{ld}}]\!] = 1$$

because the counterexample formula $\varphi(\delta_{\mathsf{ld}})$ is unsatisfiable under abstraction with three concrete processes in the spotlight. This entails $\mathsf{Car}[\![\neg\varphi(\delta_{\mathsf{ld}}) \vee \phi_{\mathsf{ld}}]\!] = 1$ by the embedding theorem 1 and thus $\digamma(\delta_{\mathsf{ld}})$ by Lemma 1, i.e. δ_{ld} is spurious.

We can not yet conclude that $\mathsf{Car}[\![\phi_{\mathsf{Id}}]\!]$ holds, obviously because there may be other counterexamples besides δ_{Id}. However, the fact that a counterexample is spurious allows us to reduce the satisfaction analysis to those runs where the counterexample formula is definitely violated. Thus by Lemma 1, we can use the counterexample formula as a source for shadow refinement simply by binding the fresh variables of the counterexample formula to $\perp$. By this, we eliminate behaviour of $\perp$ that was shown to be not possible with any concrete processes.

Lemma 2 (Shadow Refinement). *Let* D *be a dynamic system over signature* $\mathcal{S}$, $\phi \in \mathit{Specs}_{\mathcal{S}}$ *and* $\delta \in \mathit{Cex}(\mathsf{D}^{\sharp}_{X}[\![\phi]\!])$ *a counterexample with* $\digamma(\delta)$*. Then*

$$\mathsf{D}^{\sharp}_{F}[\![\varphi(\delta) \geq 1/2 \vee \phi]\!] \;\sqsubseteq\; \mathsf{D}[\![\phi]\!]$$

for $F := \mathit{fresh}(\varphi(\delta))$*.* ♢

Applied to the case study, we check

$$\mathsf{Car}^{\sharp}_{\{x_{4,1}\}}[\![\varphi(\delta_{\mathsf{Id}}) \geq 1/2 \vee \phi_{\mathsf{Id}}]\!]$$

for which we obtain the result 1, intuitively because ϕ_{Id} holds on all runs where $\varphi(\delta_{\mathsf{Id}})$ is definitely violated. From Lemma 2, we conclude that $\mathsf{Car}[\![\phi_{\mathsf{Id}}]\!] = 1$.

This procedure of validation and refinement can be iterated in a standard refinement loop, where the iteration runs as long as we obtain an indefinite result. We must however be prepared that a counterexample may not be (in-)validated via a single verification run, because checking the counterexample formula according to Lemma 1 *under spotlight abstraction* may not yield a definite answer. In this case we can use the same validation and refinement procedure for the counterexample of the counterexample formula. Algorithm 1 called $\mathsf{check}(\mathsf{D}, \phi)$ implements our idea of counterexample guided spotlight abstraction refinement by recursively calling itself for iterative counterexample validation. By the above lemmata 1 and 2, we have that $\mathsf{D}[\![\phi]\!] \iff \mathsf{check}(\mathsf{D}, \phi)$.

Algorithm 1. $\mathsf{check}(\mathsf{D}, \phi)$ returns $\mathbb{B}$

```
 1: let $F := \emptyset$
 2: let $b := \mathsf{D}^{\sharp}_{F}[\![\phi]\!]$
 3: while $b = 1/2$ do
 4:     let $\delta \in \mathit{Cex}(\mathsf{D}^{\sharp}_{F}[\![\phi]\!])$
 5:     if $\mathsf{check}(\mathsf{D}, \neg\varphi(\delta) \vee \phi)$ then
 6:         let $F := F \cup \mathit{fresh}(\varphi(\delta))$
 7:         let $\phi := \varphi(\delta) \geq 1/2 \vee \phi$
 8:         let $b := \mathsf{D}^{\sharp}_{F}[\![\phi]\!]$
 9:     else
10:         $b := 0$
11:     end if
12: end while
13: return $b$
```

As a consequence of the undecidability of the verification problem, algorithm 1 does not terminate in general. However, in each recursion depth the spotlight is enlarged and each iteration eliminates a new source for spurious interference. In fact it can be shown [20] that the only source for divergence of the algorithm is the recursive counterexample validation, whereas the iterative refinement is guaranteed to finally terminate. We thus anticipate that the overall algorithm terminates for a relevant class of dynamic systems, namely for those where counterexamples can be (in-)validated via a finite number of concrete processes. First promising experiments are given in the next Section.

5 Evaluation

We use an existing verification environment for dynamic systems, which has been developed in the course of [10], for a first experimental evaluation of our approach. The toolset comprises a compiler from XML descriptions of dynamic systems to input languages of finite-state model-checkers, and integrates the spotlight abstraction implementation of [25]. The experiments were performed by the VIS 2.1 model-checker [6] on a Linux host with 3 GHz and 2 GB of RAM. In Table 1 below, 'rec' denotes the actual recursion depth of algorithm 1, 'iter' the iteration counter in this depth, and 'spot' the maximal size of the spotlight, that is, the number of concrete processes.

The upper part of Table 1 shows the verification tasks that are necessary to verify the running example property ϕ_{ld} for the car platooning system. As a second experiment, we demonstrate that we are able to obtain *concrete* counterexamples under spotlight abstraction. This is of special importance as a typical debugging application of model-checking is to check whether a certain *desired* configuration is reachable. Therefore, one claims that the negation of the configuration is globally true and expects a counterexample. The lower part of Table 1 shows the verification tasks necessary to disprove the property $\phi_{\mathsf{fl}} := \mathsf{G}\,\neg\mathsf{fl}(x)$, that is, to find a concrete witness for a car becoming a follower car (cf. Fig. 3).

We furthermore evaluated our approach on a case study concerning a scatternet formation [26] roughly following the bluetooth connection scenario. In this protocol, mobile devices are grouped into piconets comprising one master device and a finite set of slave devices connected in a star topology. Piconets may then merge into scatternets where one slave device serves as a bridge, that is, it is

Table 1. Task flow of $\mathsf{check}(\mathsf{Car}, \phi_{\mathsf{ld}})$ and $\mathsf{check}(\mathsf{Car}, \phi_{\mathsf{fl}})$

rec	iter	task	spot	result	time	memory
0	0	$\mathsf{Car}^{\sharp}_{\emptyset}[\![\phi_{\mathsf{ld}}]\!]$	2	$1/2$ (δ_{ld})	6 s	2 MB
1	0	$\mathsf{Car}^{\sharp}_{\emptyset}[\![\neg\varphi(\delta_{\mathsf{ld}}) \vee \phi_{\mathsf{ld}}]\!]$	3	1	42 s	3 MB
1	0	return 1				
0	1	$\mathsf{Car}^{\sharp}_{\{x_{4,1}\}}[\![\varphi(\delta_{\mathsf{ld}}) \geq 1/2 \vee \phi_{\mathsf{ld}}]\!]$	2	1	8 s	2 MB
0	1	return 1				
0	0	$\mathsf{Car}^{\sharp}_{\emptyset}[\![\phi_{\mathsf{fl}}]\!]$	1	$1/2$ (δ_{fl})	3 s	2 MB
1	0	$\mathsf{Car}^{\sharp}_{\emptyset}[\![\neg\varphi(\delta) \vee \phi_{\mathsf{fl}}]\!]$	2	0 (π_{fl})	7 s	2 MB
1	0	return 0				
0	0	return 0				

a slave in two different piconets at the same time and is routing information from one piconet to the other. The case study consists of seven evolution rules, and the running times of the verification tasks are below two minutes each. We needed a recursion depth of two with a maximal spotlight size of three to prove that a device is able become a bridge device. Two iterations and a recursion depth of one allows us to verify the safety property that a pure slave device has a connection to exactly one master device.

6 Conclusion

We have presented an iterative refinement scheme for spotlight abstractions that allows us to formally verify dynamic systems against first-order temporal specifications. To the best of our knowledge, this is the first iterative refinement approach in this research direction. Although spotlight abstraction can be formulated [8] as an instance of the canonical abstraction framework [18], our results reveal a quite different nature for refinement: While predicate abstraction allows us to identify spurious counterexamples by simulation but may diverge during the refinement steps, spotlight abstraction shifts the problem into the validation of counterexamples while the refinement itself can be done very effectively.

A strong point of our approach is that we may start with a minimal number of concrete processes, and enlarge the spotlight only *gradually* driven by abstract counterexamples. In doing so we keep the number of concurrent processes as small as possible in order to avoid combinatorial explosion of the model-checking tasks. The running times of the experiments confirm the importance of this issue.

For future work, we aim at a more in-depth investigation of termination properties of our algorithm. It will be worthwhile to integrate existing techniques for shadow refinement [14,15] in order to reduce the number of iterations. Also, the application of spotlight abstraction refinement in the area of heap manipulating programs [18] is of interest (see the discussion on related work on page 24).

References

1. UNISIG: Subset 026-ch. 3; vers. 2.2.2 (srs) (March 2002), `http://www.aeif.org/ccm/default.asp`
2. Kripke, S.: Semantical Considerations on Modal Logic. Acta Phil. Fennica 16, 83–94 (1963)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
4. Kesten, Y., Pnueli, A.: Control and Data Abstraction: The Cornerstones of Practical Formal Verification. International Journal on Software Tools for Technology Transfer 2(4), 328–342 (2000)
5. Fitting, M., Mendelsohn, R.L.: First Order Modal Logic. Kluwer, Dordrecht (1998)
6. Brayton, R.K., Hachtel, G.D., Sangiovanni - Vincentelli, A.L., Somenzi, F., Aziz, A., Cheng, S.T., Edwards, S.A., Khatri, S.P., Kukimoto, Y., Pardo, A., Qadeer, S., Ranjan, R.K., Sarwary, S., Shiple, T.R., Swamy, G., Villa, T.: VIS: a System for Verification and Synthesis. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 428–432. Springer, Heidelberg (1996)

7. Holzmann, G.J.: The SPIN model checker: Primer and reference manual. Addison-Wesley, Reading (2004)
8. Wachter, B., Westphal, B.: The Spotlight Principle. On Combining Process-Summarising State Abstractions. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 182–198. Springer, Heidelberg (2007)
9. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-Guided Abstraction Refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
10. Rakow, J.: Verification of Dynamic Communication Systems. Master's thesis, Carl von Ossietzky Universität Oldenburg (April 2006)
11. Westphal, B.: LSC Verification for UML Models with Unbounded Creation and Destruction. Electr. Notes Theor. Comput. Sci. 144(3), 133–145 (2006)
12. Miller, A., Calder, M.: An automatic abstraction technique for verifying featured, parameterised systems. Theor. Comput. Sci. (to appear, 2007)
13. Damm, W., Westphal, B.: Live and let die: LSC-based verification of UML-models. Science of Computer Programming 55(1–3), 117–159 (2005)
14. Toben, T.: Non-Interference Properties for Data-Type Reduction of Communicating Systems. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 619–638. Springer, Heidelberg (2007)
15. Bauer, J., Toben, T., Westphal, B.: Mind the Shapes: Abstraction Refinement via Topology Invariants. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 35–50. Springer, Heidelberg (2007)
16. Bauer, J., Wilhelm, R.: Static Analysis of Dynamic Communication Systems by Partner Abstraction. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 249–264. Springer, Heidelberg (2007)
17. König, B., Kozioura, V.: Counterexample-Guided Abstraction Refinement for the Analysis of Graph Transformation Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 197–211. Springer, Heidelberg (2006)
18. Sagiv, S., Reps, T.W., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ACM Trans. Program. Lang. Syst. 24(3), 217–298 (2002)
19. Kleene, S.C.: Introduction to metamathematics. Bibl. Matematica. North-Holland, Amsterdam (1952)
20. Toben, T.: Spotlight Abstraction Refinement by Evolution Constraints. PhD thesis, Carl von Ossietzky Universität Oldenburg (to appear, 2008)
21. Pnueli, A.: The temporal logic of programs. In: Proc. FOCS, pp. 46–57. IEEE, Los Alamitos (1977)
22. Westphal, B.: Specification and Verification of Dynamic Topology Systems. PhD thesis, Carl von Ossietzky Universität Oldenburg (2008)
23. Xie, F., Browne, J.C.: Integrated State Space Reduction for Model Checking Executable Object-oriented Software System Designs. In: Kutsche, R.-D., Weber, H. (eds.) ETAPS 2002 and FASE 2002. LNCS, vol. 2306, pp. 64–79. Springer, Heidelberg (2002)
24. Vardi, M.Y., Wolper, P.: An Automata-Theoretic Approach to Automatic Program Verification. In: Proc. LICS 1986, pp. 332–344. IEEE Computer Society, Los Alamitos (1986)
25. Westphal, B., Cook, B.S.: LSC Verification for UML Models with Unbounded Creation and Destruction. In: B. Cook, S., Stoller, W.V. (eds.) Proc. SoftMC 2005. ENTCS, vol. 144(3), pp. 133–145. Elsevier B.V, Amsterdam (2005)
26. Haartsen, J.: Bluetooth – the universal radio interface for adhoc, wireless connectivity. Ericsson Review 3 (1998)