



---

AVACS – Automatic Verification and Analysis of  
Complex Systems

REPORTS  
of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

---

The Temporal Logic of Appearance and  
Disappearance

by  
Jörg Bauer, Tobe Toben and Bernd Westphal

---

AVACS Technical Report No. 24  
June 2007  
ISSN: 1860-9821

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

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

# The Temporal Logic of Appearance and Disappearance

Jörg Bauer<sup>1</sup>, Tobe Toben<sup>2</sup> and Bernd Westphal<sup>2</sup>

<sup>1</sup> Technical University of Denmark, Kongens Lyngby, Denmark, joba@imm.dtu.dk\*

<sup>2</sup> Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany,  
{toben,westphal}@informatik.uni-oldenburg.de\*\*

**Abstract.** An interesting class of systems is characterised by a changing and potentially unbounded number of processes. Prominent instances are ad-hoc networks, dynamic communication systems, and object-oriented programs. Properties of such systems are naturally specified by predicative variants of temporal logic. So a number of logics have been proposed. However, the proposals are inconsistent among each other and lack a common understanding of the underlying systems.

In this article, we comprehensively identify features whose characteristics determine the design of such logics. Given dynamic object creation and destruction, it turns out that the most crucial feature is that of *disappearance*. That is, how does a formula evaluate if its actors disappear? While some approaches avoid this problem at all, we propose a generalised logic subsuming the existing ones. We show that our proposal conservatively extends propositional temporal logic and conduct first investigations of emerging new issues of such temporal logics.

## 1 Introduction

Verification of temporal properties for *parameterised systems* is a well-established topic. Parameterised systems typically consist of an arbitrary, but fixed number of processes that may interact via shared memory or message passing. Properties of parameterised systems are typically specified by classical temporal logics augmented with quantification over the processes.

The problem of specification (and verification) becomes considerably more complex if the number of processes may *change* during run-time, that is, if processes are appearing and disappearing without a known upper bound on their number. For each temporal formula covering more than a single state we have to cater for the case that processes (or objects) referred to *now* may have disappeared already after the next system step.

This latter situation occurs, e.g., in the verification of Java or heap-manipulating programs, object-oriented models, or dynamic communication systems (DCS) [1]. In these domains, formal verification tools have been developed accompanied by temporal specification logics tailored to the analyses.

\* Partly sponsored by project SENSORIA, IST-2005-016004, and by a visiting researcher grant of SFB/TR 14 AVACS.

\*\* Partly supported by German Research Council (DFG) within SFB/TR 14 AVACS.

*The Existing Proposals.* The most basic approach has been taken by the Bandera Specification Logic (BSL) [2], a collection of LTL-based patterns for Java verification in the Bandera tool. The strategy is to simply avoid problems. Disappearance is treated by explicitly excluding it from the semantical domain: objects may appear but are expected to live forever. Evaluation of a quantified formula only starts after sufficiently many objects have appeared. Similarly, the dynamic logic ODL [3] currently only considers creation.

A completely different aim is targeted by Allocational Temporal Logic (ATL) [4] (and its variant NallTL). The intention is to analyse memory allocation and deallocation properties of heap-manipulating programs. Consequently, appearance, aliveness, and disappearance are treated explicitly. However, they are also the only considered properties of objects.

In the domain of object-orientation, a large body of literature proposes to extend UML’s (non-temporal) Object Constraint Language (OCL) [5] with temporal operators. Distefano et al. [4, 6] propose to replace OCL by, basically, a merger of ATL and OCL called BOTL.<sup>3</sup> Semantically, they aim for what we’ll call *witness semantics*, that is, they focus on the life-time of objects such that a “globally” property is satisfied if it holds during life-time. Unfortunately, they don’t discuss the crucial “next” operator in sufficient detail. The focus of Cengarle et al. [7] are real-time properties, so they equip events with time-stamps and sketch how temporal properties reduce to time-stamps. Ziemann et al. [8] straightforwardly propose to use OCL as the term language for first-order LTL, but remain on a sketchy level concerning semantics. Similarly, Bradfield et al. [9] propose  $O\mu$ , which uses OCL as the term language of a  $\mu$ -calculus variant.

Most notably, all of these proposals don’t discuss the problem of disappearance explicitly, and all of them implicitly opt for what we’ll call a *biased semantics*. That is, they explicitly define the positive case and map both, a negative witness disproving a formula *and* pre-mature disappearance of objects to “false”, although OCL provides the third logical value *oclUndefined*. Only Flake et al. [10, 11] employ this value in their proposal, OCL as term language of LTL with past operators. Unfortunately, they don’t discuss their intention for treating disappearance; their definitions seem technically flawed as they miss some border-cases. Also in the context of UML, but independent from OCL, the scenario language of LSCs [12, 13] has been adopted to systems with dynamic appearance and disappearance of objects. Creation and destruction of objects become explicit by the LSCs’ life-lines: a specification is violated if an object disappears before the end of its life-line.

More recently, Evolution Temporal Logic (VTL, formerly called ETL) has been proposed by Yahav et al. [14, 15] in the context of the abstract interpretation-based analysis of Java-like, heap-manipulating programs. They are the first to openly discuss some of their design decisions, for instance, disjoint universe (cf. Sect. 2). The logic METT [1] is basically VTL tailored to DCS by removing transitive closure and adding operators to refer to event-based communication.

---

<sup>3</sup> In the following, BOTL denotes the linear variant [4], not the branching one [6].

*The Problem.* Interestingly, over time there is a syntax-wise convergence towards VTL. It abstracts from concrete term languages like OCL by a set of unary and binary predicates and joins most of the typical syntactic constructs into LTL with quantification over processes and *aliveness primitives*. They refer to whether an object is *newborn* (“ $\odot$ ”), *alive* (“ $\ominus$ ”), or *disappearing* (“ $\otimes$ ”). However, there is no sign of convergence with respect to the *semantics*. As a striking example for the same property (there exists a process *now*, which will not be newborn after the *next* evolution step) evaluating differently consider the following formula:

$$\varphi_0 = \exists x . X \neg \odot x. \tag{1}$$

Now let  $\pi$  be a computation path where there is one process in the initial state, which disappears in the first step. According to ATL,  $\varphi_0$  evaluates to true over  $\pi$ , while it evaluates to false in VTL. Recall that BSL doesn’t apply at all because its semantic domain explicitly *excludes* disappearance.

One *may argue* whether or not a property holding in one state shall continue to hold when the objects making it true disappear. Conversely, if we require that something will finally become true, is it allowed that a corresponding object only appears in the future? The aforementioned logics have (legally) fixed their own solutions to the problem of disappearance. Our critique addresses the frequent lack of motivation, discussion, or making explicit their particular choices. They simply do *not argue*.

The combination of dynamic universes and modalities has already been discussed half a century ago in philosophical logic. The famous *Barcan formula* [16],  $\forall x . \mathbf{G} p(x) \rightarrow \mathbf{G} \forall x . p(x)$ , relates closely to the problem of disappearance. Kripke [17] and Lewis [18] discussed, whether dynamically evolving universes should have the same or disjoint elements. Unfortunately, none of the related approaches refers this work with the notable exception of [15], where a choice is made in favour of Lewis’ disjoint universes.

*The Plan.* Our plan for the paper is as follows. In Sect. 2, we identify (presumably for the first time) a comprehensive set of *features* related to the problem of disappearance. Their characteristics explicate the choices underlying the design of specification logics for systems with a changing and potentially unbounded number of objects. To aid the design of future logics, we amplify how these features are interrelated and influence each other. Based on the most complex characteristics, we define the new, generalised logic DOCTL\* in Sect. 3. As DOCTL\* subsumes existing proposals, we can formally compare related logics. Section 4 states some initial properties of DOCTL\* and Sect. 5 concludes.

## 2 Dimensions of the Problem of Disappearance

A logic can be viewed to be determined by four constituents:

$$\begin{array}{ccc} \text{(syntax)} & \mathcal{S} & \\ \text{(semantical domain)} & \mathcal{U} & \end{array} \begin{array}{c} \searrow \\ \searrow \end{array} \llbracket \cdot \rrbracket \longrightarrow \mathcal{B} \quad \text{(logical domain)}$$

The *syntax*  $\mathcal{S}$  specifies the set of formulae and the *semantical domain*  $\mathcal{U}$  specifies, where formulae are evaluated *on*. The semantical domain of LTL, e.g., may be a set of computation paths or a transition system. The *logical domain*  $\mathcal{B}$ , that is, where formulae evaluate *to*, is often boolean but may be multi-valued as well. Finally, the *semantical function*  $\llbracket \cdot \rrbracket$  says *how* to evaluate formulae.

When defining a logic for possibly disappearing objects, the four constituents are determined by a set of *features*, each with a set of *characteristics*, which we introduce in the following. An example feature concerning the syntax is the “next” operator, the feature’s characteristics are absence and presence. For most characteristics, we provide potential applications justifying their existence. Moreover, we allude to the characteristics of existing logics. We present each feature graphically as a *slider*, where characteristics are positions. A gray spot on a slider denotes the choice we made for DOCTL\* (cf. Sect. 3).

With each feature, we point out existing interdependencies if present: choosing a characteristic of one feature may limit the choice for others. In-depth elaboration on these dependencies is not in the scope of this paper, though.

## 2.1 Semantical Domain.

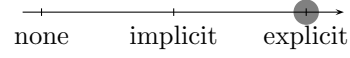
We assume *transition systems* as semantical domain because DOCTL\* will be a branching time logic. This covers in particular logics over computation paths. While this is fixed, the particular choice of the *states* of the transition system is still open. We’ll call these states *topologies* and their formal representation *universes*.

*Universe.* This slider relates to the discussion between Kripke [17] and Lewis [18] about evolving universes, that is, *fixed universe* versus *counterpart* semantics. In the counterpart semantics, the concept of identities is by definition local to topologies. One cannot state that a newly created object will obtain the identity *id* and keep it in a future state. As an identity exists in at most one topology, there is a need to explicitly relate the identities in two topologies that represent the same object by an *evolution relation*. Based on this relation, one can define equality and refer to the lifetime of an individual. In a fixed universe setting, the concept of identity is global. The same identity may occur in different topologies. We can further distinguish “fixed” where all states share exactly the same universe and “variable” where states are (not necessarily disjoint) subsets of a global universe.

We have deliberately *not* put a spot in this slider, because DOCTL\* is parametric in this choice (cf. Sect. 3). The VTL authors explicitly advocate a disjoint universe motivated by their application domain, while (variants of) ATL pick a middle slider position without (explicitly) motivating it. Classical parameterised systems are naturally of the fixed/fixed kind.

When considering object-oriented programs, the “disjoint” position lends itself to abstract from physical memory addresses. This is somewhat similar to the *storeless semantics* [19] concept. In contrast, when considering the whole memory to be there all the time a leftmost slider position seems more appropriate.

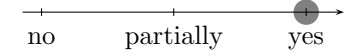
*Indication of Evolution.* The position of this slider indicates whether the semantical domain provides means to reason about the evolution of individuals. With any choice of universe, one may decide not to consider evolution at all or to explicitly provide an evolution relation. With a fixed universe, one can interpret the shared identities to implicitly indicate evolution.



DOCTL\* assumes an evolution relation relating individuals in one topology to their instances in successor topologies. As a notable variant of “explicit”, VTL considers computation paths where successor states are annotated by sets of objects that appear/disappear during one transition.

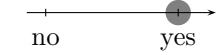
If one is interested in non-temporal properties, like absence of memory-leaks, the left position is fine. Otherwise one should rather opt for tracking evolution.

*Life-Cycle.* This slider determines whether the semantical domain has a concept of individuals being newborn, alive, or dying. Such a concept isn’t necessary, when only referring to alive objects. The slider position is principally independent of the universe and evolution sliders. Even when not tracking evolution, one may know (by annotation) that some objects are new in this state.

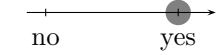


The only logic other than DOCTL\* with the spot on “yes” is ATL because it genuinely addresses object allocation. VTL has the spot on “partially” as it artificially extends the life-time of objects to indicate death.

*Disappearance.* This slider is on “yes” if individuals can actually disappear in the semantic domain. Note that this slider may be on “no”, although there is a notion of life-cycle. The semantical domain of BSL, e.g., has a notion of “birth” but doesn’t permit disappearance.



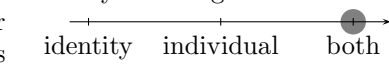
*Re-use.* Is it possible, in the semantical domain, that one individual disappears and re-appears later? If one is interested in a more abstract view of, e.g., a heap, then the “no” position is sufficient. If one wants to reason about memory allocation policies or systems, where individuals have fixed “names” (IP addresses, phone numbers), it is a good advice to opt for re-use. Again, storeless semantics [19] is a prominent use-case for the left slider position. Next to the object-oriented domain, DOCTL\* is the only logic considering re-use. Other approaches either avoid to mention re-use at all or simply exclude it by assuming a “clean” semantical domain.



*Dangling References.* Is it possible that there are relations between alive and non-alive individuals? In heap examples this corresponds to dangling references, and in DCS [1], a process may still know the identity of an already deceased process. The most refined choice will be able to distinguish “segmentation faults” (accesses to non-allocated memory cells) and erroneous but valid access to freed memory. Variants of OCL naturally consider this case, while VTL for example considers only systems with garbage-collection semantics, where disappearance depends on absence of links.

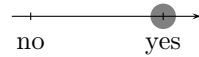


## 2.2 Syntax

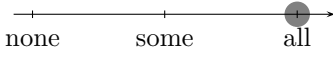
*Quantification.* In the domain of classical parameterised systems logical variables clearly denote process identities. With appearance and disappearance, in contrast, there is a *choice*. Firstly, logical variables may also range over identities. Then we have to be prepared to encounter  currently unused identities: logical variables may be dangling references. Alternatively, logical variables may only consider *alive* individuals and, if evolution is traced, their (possibly finite) destiny. That is, the evaluation of temporal formulae is restricted to the life-time of objects. Technically, the life-time is explicitly constructed during formula evaluation [10] or implicitly by a notion of stepwise assignment evolution [14]. We call this a *witness semantics*, since there must be one alive positive or negative witness to make a formula true or false. Disappearance then still poses a problem if the “next” operator is part of the logic (cf. Sect. 4). Note that both notions coincide for classical parameterised systems. As all processes are constantly alive, an identity denotes a destiny and vice versa.

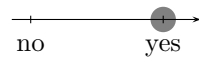
Quantification over identities may be interesting if one considers garbage as dead individuals and wants to quantify over it, e.g., to state that a particular heap cell will finally host an object. In this view, “individual” reasons over non-garbage only and “identity” over all memory cells. “Individual” is a natural choice if the universe is “disjoint”.

To the best of our knowledge, this slider is a new insight. Each of the existing approaches makes its choice, without discussion. Only for VTL the choice is determined by the employed disjoint universe.

*Next.* Having a next operator or not is a well-known issue  in temporal logics. In the presence of disappearance, it is highly relevant because it poses the problem to explain what it means to access the next state of an object which is present *now* but gone in the next state.

Interestingly, all proposals named in Sect. 1 consider “next” to be a necessary temporal operator, but don’t discuss its issues. In Sect. 4 we’ll see why a combination of witness semantics and omitting “next” is a sensible option.

*Life-Cycle Queries.* If the semantical domain  has a notion of life-cycle (cf. Sect. 2.1), the latter can be accessible by syntax or not. Existing logics’ slider position corresponds closely to the semantical life-cycle slider. BSL’s semantical concept of birth, however, is not accessible in the syntax.

*Additional Operators and Equality.* This slider denotes  whether there are other operators in the syntax, which are not captured by any of the sliders yet. Transitive closure, for example, is crucial for VTL, while NallTL provides notions for heap navigation. DOCTL\* doesn’t provide any additional operators except for comparison for equality.



*Topology-labelled Transition System (TLTS).* A *topology-labelled transitions system* over a set of identities  $Id$  and predicate symbols  $\mathcal{P}$  is a quadruple  $M = (S, S_0, (R, e), \mathcal{L})$  consisting of a (possibly infinite) set of states  $S$ , initial states  $S_0 \subseteq S$ , a transition relation  $R \subseteq S \times S$ , and a labelling function  $\mathcal{L}$ , which assigns each state from  $S$  a topology over  $Id$  and  $\mathcal{P}$ . Figure 1(b) to 1(d) indicate how to encode the three universe characteristics. Evolution is traced by an *evolution function*  $e$ , which assigns each transition  $r = (s, s') \in R$  a partial function  $e_r : \odot(\mathcal{L}(s)) \rightarrow \odot(\mathcal{L}(s'))$ .

The definition of *computation path* in  $M$  is standard. A computation path in  $M$  starting at state  $s_0$  is an (infinite) sequence  $\pi = s_0, s_1, \dots$  such that  $(s_i, s_{i+1}) \in R$ . We use  $\pi^k$  to denote the  $k$ -th element of  $\pi$  and  $\pi/k$  to denote the suffix of  $\pi$  beginning with the  $k$ -th element. The paths in  $M$  starting at  $s$  are denoted by  $\Pi_M(s)$ , the paths starting in an initial state by  $\Pi_M$ .

Given a computation path  $\pi = s_0, s_1, \dots$ , a (finite or infinite) sequence  $\delta = \langle id_0, id_1, id_2, \dots \rangle$  of identities is called *evolution chain* of  $id_0$  along  $\pi$  if and only if the identities are alive and consecutive identities evolve into each other according to the evolution function of the corresponding transition, i.e., if  $id_i \in \odot(\mathcal{L}(s_i))$  and  $(id_i, id_{i+1}) \in e_{(s_i, s_{i+1})}$ ,  $i \geq 0$ .

We use  $\langle \rangle$  to denote the empty sequence. Like for computation paths,  $\delta^k$  denotes the  $k$ -th element of sequence  $\delta$ , and  $\delta/k$  the suffix of  $\delta$  starting with the  $k$ -th element. The latter is empty if the length of  $\delta$  is smaller than  $k$ .

*DOCTL\**. Let  $V = V^T \dot{\cup} V^{\mathbf{T}}$  be a set of logical variables of two types, *identity variables*  $x, y, \dots \in V^T$  of type  $T$  denoting identities and *destiny variables*  $\mathbf{x}, \mathbf{y}, \dots \in V^{\mathbf{T}}$  of type  $\mathbf{T}$  denoting evolution chains.

*State and path formulae* of DOCTL\* are given by grammars (2) and (3).<sup>4</sup>

$$\phi ::= 1 \mid \odot v_1 \mid v_1 = v_2 \mid p(v_1, \dots, v_k) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \forall x . \phi \mid \mathbf{A} \psi, \quad (2)$$

$$\psi ::= \phi \mid \odot v_1 \mid \otimes v_1 \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \forall \mathbf{x} . \psi \mid \mathbf{X} \psi \mid \mathbf{G} \psi \mid \mathbf{F} \psi \mid \psi_1 \mathbf{U} \psi_2, \quad (3)$$

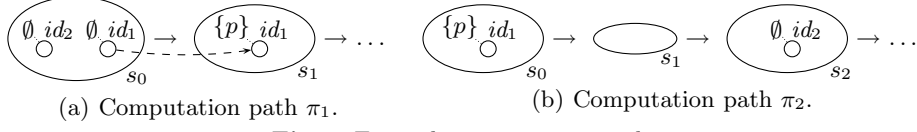
where  $v_i \in V$  is a logical variable of either type and  $p \in \mathcal{P}$  a predicate symbol. Logical variables are *bound* by  $\forall$  only. We use  $free(\psi)$  to denote the set of logical variables occurring free in  $\psi$ .

Let  $V_0 \subseteq V$  be a set of logical variables. An *assignment* of  $V_0$  in a path  $\pi$  is a function  $\theta$  assigning each  $x \in V_0^T$  an identity from  $Id$  and each  $\mathbf{x} \in V_0^{\mathbf{T}}$  an evolution chain along  $\pi$ . We use  $\theta[v \mapsto d]$  to denote the modification of  $\theta$  at  $v$ , that is, the assignment yielding  $d$  for  $v$  and coinciding with  $\theta$  otherwise. We use  $\theta/k$  to denote  $\theta[\mathbf{x}_1 \mapsto \theta(\mathbf{x}_1)/k] \dots [\mathbf{x}_n \mapsto \theta(\mathbf{x}_n)/k]$ , where  $\mathbf{x}_1, \dots, \mathbf{x}_n \in V_0^{\mathbf{T}}$ . For notational convenience, we set  $\theta(x)^0 := \theta(x)$  for not sequence-valued variables.

In contrast to the biased logics, which define the satisfaction relation between formulae and transition systems directly, we define the *evaluation* of a formula. Given a computation path  $\pi$  in TLTS  $M$ , the evaluation over  $\pi$  at  $n \in \mathbb{N}_0$  under an assignment  $\theta$  of  $free(\phi)$  in  $\pi$  is defined inductively as follows.

<sup>4</sup> Hence for the logic, individuals are *anonymous* as  $Id$  is completely uninterpreted, in particular unordered, and the syntax provides at most comparison of identities.





**Fig. 2.** Example computation paths.

*Definiteness.* From the motivation of DOCTL\*, the indefinite truth value  $1/2$  is intended to occur in exceptional situations like pre-mature disappearance. An interesting question is whether we can syntactically characterise formulae that are *robust* against exceptional situations, in other words, that always evaluate definite. The following theorem presents a sufficient condition for definiteness.

**Theorem 1 ([22]).** *Given a computation path  $\pi$  in  $M$  and an assignment  $\theta$  of the free variables of DOCTL\* formula  $\varphi$ , the formula evaluates definite if*

1.  $\varphi$  is quantifier-free and all variables are destiny-variables, i.e. of type  $\mathbf{T}$ ,
2. there is no sub-formula of the form  $\mathbf{X}\psi$ ,
3. all predicates from  $\mathcal{P}$  evaluate definite along  $\pi$ , and
4. for each sub-formula  $\psi_1 \mathbf{U} \psi_2$ , the individuals referred to in  $\psi_1$  (via  $\theta$ ) live at least as long as the ones referred to in  $\psi_2$  along  $\pi$ .  $\diamond$

The right formula of (4) meets the premises and thus evaluates definite for the (suffixes of) computation paths  $\pi_1$  and  $\pi_2$ . In contrast, the formula  $\mathbf{X}p(\mathbf{x})$  violates requirement (2.) and indeed evaluates to  $1/2$  for  $\pi_1$ .

Theorem 1 has the nice corollary that formulae over destiny variables with only  $\mathbf{G}$  and  $\mathbf{F}$  evaluate definite if the first three criteria are met. The reason is obvious from the left law in (4). Namely that the left hand side of  $\mathbf{U}$  doesn't refer to any individuals. Furthermore, we see that if a logic neither considers disappearance nor the possibility of predicates being indefinite, then all formulae always evaluate definite. Both premises hold for BSL, for instance.

*Further Remarks.* We conclude this section by some further observations about DOCTL\*. It will be interesting to see how slider positions affect properties in general, e.g., the Barcan equivalence doesn't generally hold for variable universes.

The reader may have recognised that we didn't motivate the *necessity* of destiny variables, i.e., that the “quantification” slider has three positions. By assuming both, the subsumption of all other approaches becomes evident. Closer inspection shows that identity variables can mimic destiny variables if we have the disappearance operator  $\otimes$ . For example,  $\forall \mathbf{x} . p_1(\mathbf{x}) \mathbf{U} p_2(\mathbf{x})$  is equivalent to  $\forall x . p_1(x) \mathbf{U} (\otimes x \vee p_2(x))$ . That is, end of consideration at the end of life-time is made explicit. The general case is of course more intricate.

A second issue concerns prenex normal forms. We conjecture that they exist for identity variables, while they do not for destiny variables, as their binding depends on the individuals in the topology where they're bound.

Thirdly, we can state that DOCTL\* provides more than the union of the existing approaches. For example, we obtain more differentiated answers for typical temporal queries employing  $\mathbf{F}$  or  $\mathbf{G}$ . Consider the computation path  $\pi_2$

from Fig. 2(b) and formula  $\exists x. \mathbf{G} p(x)$ . This query is answered negative by both VTL and ATL for any suffix of  $\pi_2$ , but with different reasons for different suffixes. Namely, the formula fails for (i)  $\pi_2$  because there is no evolution counterpart of  $id_1$  in  $s_1$ , (ii)  $\pi_2/1$  because topology  $s_1$  is empty, thus the quantification is trivially 0, and (iii)  $\pi_2/j, j \geq 2$ , because  $p(id_j)$  doesn't hold. With DOCTL\*, we obtain the following answers, depending on the type of quantified variable.

$(v, j)$	$(x, 0)$	$(x, 1)$	$(x, \geq 2)$	$(\mathbf{x}, 0)$	$(\mathbf{x}, 1)$	$(\mathbf{x}, \geq 2)$
$\llbracket \exists v. \mathbf{G} p(v) \rrbracket (\pi_2, j, \theta)$	1/2	1/2	0	1	0	0

Finally we observe, not surprisingly, DOCTL\* does not enjoy the *finite model property*, that is, a satisfiable DOCTL\* formula is not necessarily satisfiable in a finite system. For example, the requirement that in each step a new entity is created while the existing ones do not disappear, i.e.,  $\mathbf{G}(\forall x. (\odot x \rightarrow \neg \otimes x) \wedge \exists y. \odot y)$  is only satisfiable in an infinite transition system.

## 5 Conclusion and Further Work

We have identified the problem of disappearance to be crucial when specifying properties of systems with a dynamically changing and potentially unbounded number of objects or processes. We discovered a number of anomalies and inconsistencies both within and among existing approaches, whose treatment of the problem ranges from avoiding it at all to biased evaluations. Our advice is to thoroughly discuss the problem and to avoid (implicit) biasing. To support such discussions, we make explicit the choices a logics designer has to face.

Based on these choices we have defined DOCTL\*, a generalisation of existing approaches. This is presumably the first time such a logic was defined fully-aware of all subtleties and under reflection of design decisions – rather than in an ad-hoc manner. Another benefit of our feature taxonomy is that all existing approaches can be surveyed and compared in a concise and formal fashion (cf. App. A). Even if DOCTL\* turns out to be too expressive for certain purposes, it can still serve as a valuable starting point for the design of less expressive fragments.

Taking a broader perspective, we benefit from philosophical work of Barcan [16], Lewis [18], and Kripke [17] in a way similar to how formal methods benefited from modal logic. Modal logic had long been discussed in philosophical logic before it was identified as highly relevant for the reactive systems branch of computer science by Pnueli [23]. Now it seems that the aforementioned works of philosophical logic on changing universes are similarly relevant for computer science when considering the systems investigated in this article.

*Further Work.* So far, we have only scratched the surface of the field of specification of a very complex class of systems that lies beyond parameterised systems. Some theorems have been proven, but a lot of work needs to be done concerning expressiveness, definiteness, usability for verification, special cases (possibly characterised by extremal slider positions) and so on. Finally, we want to study the automatic generation of fragments depending on slider positions. We hope that theorems for the most general DOCTL\* will easily yield nice specialisations.

## References

1. Bauer, J., Schaefer, I., Toben, T., Westphal, B.: Specification and verification of Dynamic Communication Systems. In: Proc. ACSD'06, IEEE CS (2006) 189–200
2. Corbett, J.C., Dwyer, M.B., Hatchiff, J., Robby: Expressing checkable properties of dynamic systems: the Bandera specification language. *STTT* **4** (2002) 34–56
3. Beckert, B., Platzer, A.: Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In Furbach, U., Shankar, N., eds.: Proc. IJCAR'06. Number 4130 in LNCS, Springer (2006) 266–280
4. Distefano, D.: On Model Checking the Dynamics of Object-based Software. PhD thesis, University of Twente (2003)
5. OMG: Object Constraint Language, 2.0. Technical Report formal/06-05-01 (2006)
6. Distefano, D., Katoen, J.P., Rensink, A.: On a temporal logic for object-based systems. In Smith, S.F., Talcott, C.L., eds.: Proc. FMOODS'00, Kluwer (2000)
7. Cengarle, M.V., Knapp, A.: Towards OCL/RT. In Eriksson, L.H., Lindsay, P.A., eds.: Proc. FME'02. Volume 2391 of LNCS., Springer (2002) 390–409
8. Ziemann, P., Gogolla, M.: OCL extended with temporal logic. In Broy, M., Zamulin, A.V., eds.: Proc. PSI'03. Volume 2890 of LNCS., Springer (2003) 351–357
9. Bradfield, J., Filipe, J.K., Stevens, P.: Enriching OCL using observational mu-calculus. In Kutsche, R.D., Weber, H., eds.: Proc. FASE'02. Number 2306 in LNCS, Springer (2002) 203–217
10. Flake, S., Müller, W.: Formal semantics of static and temporal state-oriented OCL constraints. *Software and Systems Modeling* **2** (2003) 164–186
11. Flake, S., Müller, W.: Past- and future-oriented time-bounded temporal properties with OCL. In: Proc. SEFM'04, IEEE CS (2004) 154–163
12. Damm, W., Harel, D.: LSCs: Breathing life into Message Sequence Charts. *FMSD* **19** (2001) 45–80
13. Damm, W., Westphal, B.: Live and let die: LSC-based verification of UML-models. *Science of Computer Programming* **55** (2005) 117–159
14. Yahav, E., Reps, T., Sagiv, M., Wilhelm, R.: Verifying temporal heap properties specified via evolution logic. *Logic J. of the IGPL* **14** (2006) 755–783
15. Yahav, E., Reps, T., Sagiv, S., Wilhelm, R.: Verifying temporal heap properties specified via evolution logic. In Degano, P., ed.: Proc. ESOP'03. Number 2618 in LNCS, Springer (2003) 204–222
16. Barcan, R.C.: A functional calculus of first order based on strict implication. *J. of Symbolic Logic* **11** (1946) 1–16
17. Kripke, S.: Semantical considerations on modal logic. *Acta Phil. Fennica* **16** (1963) 83–94
18. Lewis, D.: Counterpart theory and quantified modal logic. *J. of Philosophy* **LXV** (1968) 113–126
19. Jonkers, H.: Abstract storage structures. In de Bakker, van Vliet, eds.: *Algorithmic Languages*, IFIP, North Holland (1981) 321–343
20. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. *ToPLaS* **24** (2002) 217–298
21. Kleene, S.C.: *Introduction to Metamathematics*. second edn. North-Holland (1987)
22. Westphal, B.: Specification and Analysis of Dynamic Topology Systems. PhD thesis, Carl-von-Ossietzky Universität Oldenburg (2007) to appear.
23. Pnueli, A.: The temporal logic of programs. In: Proc. FOCS'77, IEEE (1977) 46–57

## A Logics of Disappearance in Comparison

The features and characteristics from Sect. 2 allow us to survey all logics discussed in Sect. 1 on a single page by spider diagrams (cf. Fig. 3). In the following, we recall the most notable aspects from Sect. 1 and locate them in Fig. 3.

*“Slider Left”*. In contrast to DOCTL\*, one could consider the fragment where all sliders are in the left-most position (not in Fig. 3). It would not have X, life-cycle indication neither in the model nor in the logic, and not trace evolution.

*BSL* is different from that minimal logic in that it traces evolution and supports at least some notion of life-cycle. The unique feature of BSL is the choice not to consider disappearance at all (cf. Fig. 3(a)).

*ATL* is special in focusing on the life-cycle, supporting it in the computational model and in the logic. Other object properties aren’t considered (cf. Fig. 3(b)).

*NallTL* is basically a variant of ATL which in addition features a concept of topology, that is, interconnection between objects via a single link. There are consequently operators to navigate the topology in the logic (cf. Fig. 3(c)).

*BOTL* is a proposal to unify the temporal part of NallTL and ATL using OCL for state properties. It puts less strong focus on a clear life-cycle concept, but inherits the rich operator language of OCL (cf. Fig. 3(d)).

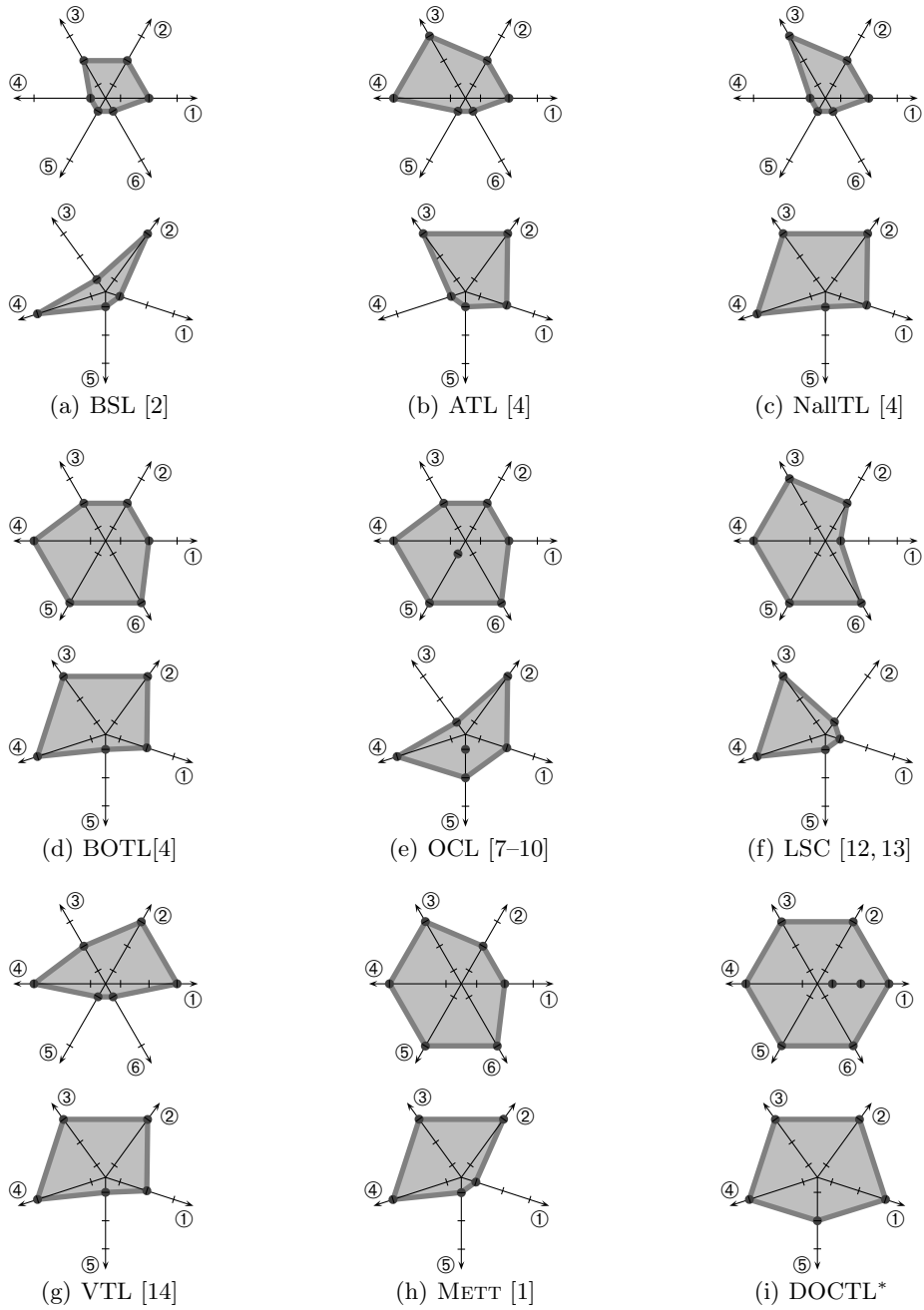
*OCL*. Diagram Fig. 3(e) covers all proposals related to OCL discussed in Sect. 1, consequently we may have multiple dots on single axes. Notable is that they’re rather homogeneous, except for the consideration of re-use and the logical domain. The only approach intending to use the three-valued domain of OCL for the temporal part is [10]. We only admit a middle slider position concerning life-cycle when this issue is not discussed, but taken for granted from UML.

*LSC*. Notable about LSC is the employment of quantification over identities as METT. Our classification of X-free is debatable as timing intervals principally allow to refer to exactly the next state in a discrete time model, but it is implicitly guarded by the life-line requiring aliveness of the object. That is, the only notion similar to X in LSCs is not of the evil character than in, e.g., VTL (cf. Fig. 3(f)).

*VTL*. VTL can be seen as a generalisation of NallTL. Next to DOCTL\*, only VTL considers disjoint universes and explicitly traces evolution. Re-use and dangling references are explicitly excluded, which is reasonable as they consider Java programs with garbage-collection. We classify the logical domain as two-valued because three-valued logic is only employed for evaluation on abstract traces (cf. Fig. 3(g)).

*METT*. Syntactically, METT is similar to VTL but it employs a different semantical domain. Namely, it doesn’t assume disjoint universes and considers variables to bind to identities rather than destinies (cf. Fig. 3(h)).

*DOCTL\** is an instance of the right-most positions for all but the logical domain slider. Recall from Sect. 2 that we could imagine some use in more than three logical values, but haven’t a clear idea of a reasonable choice yet (cf. Fig. 3(i)).



**Fig. 3. All discussed logics at a glance.** In the top diagrams: ① Universe ② Evolution ③ Life-Cycle ④ Disappearance ⑤ Re-use ⑥ Dangling References. In the bottom diagrams: ① Quantification ② Next ③ Life-Cycle ④ Operators ⑤ Logical Values.