The Spotlight Principle *

On Combining Process-Summarizing State Abstractions.

Björn Wachter[1] and Bernd Westphal[2]

[1] Universität des Saarlandes, Im Stadtwald, 66041 Saarbrücken, Germany
bwachter@cs.uni-sb.de
[2] Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
westphal@informatik.uni-oldenburg.de

Abstract. Formal verification of safety and liveness properties of systems with a dynamically changing, unbounded number of interlinked processes and infinite-domain local data is challenging due to the two sources of infiniteness. The existing state abstraction-based approaches Data Type Reduction and Environment Abstraction each address one aspect, but the former doesn't support infinite-domain local data and the latter doesn't support links and is restricted to particular properties.

The contribution of this paper is a combination of both which is obtained by first stating them in the framework of Canonical Abstraction. This new use of Canonical Abstraction, originally designed and used for the analysis of programs with heap-allocated data structures, furthermore unveils a formal connection between the two rather ad-hoc techniques.

1 Introduction

A good example for the systems we consider is car platooning as studied in the PATH project [1]. Its objective is to improve highway capacity and fuel consumption by having cars dynamically negotiate, via radio-based communication, the formation of platoons in which cars drive with reduced safety distance. A platoon consists of one or more followers and a leader, which is in particular responsible for notifying its followers in advance about braking manoeuvres. Roadside controllers announce the maximum platoon length for a certain highway segment and keep track of highway utilisation (cf. Figure 1(a)).

A formal model of the snapshot of car-platooning shown in Figure 1(a) is depicted in Figure 1(b). There, each car has a local state, like being a follower ('*flw*') or leader ('*ldr*'), and a finite-domain variable d indicating the destination, one of finitely many highway exits. A roadside controller also has a state pc, some finite-domain highway parameter x, for instance a maximum platoon length, and some infinite-domain ones, like a real-valued current utilisation of the highway y. Cars and roadside controllers do not have a global view on the entire highway, i.e. there is no shared memory, however cars have *links* to particular other cars.

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center SFB/TR 14 AVACS.

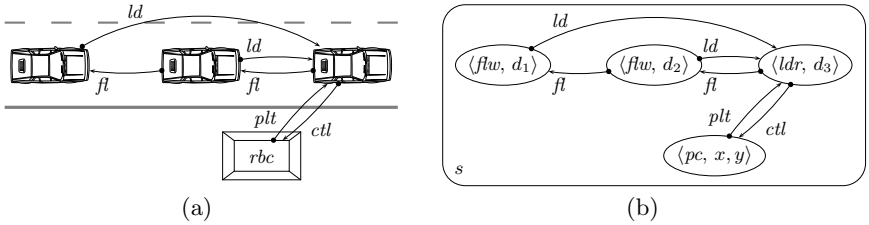


Fig. 1. Car Platooning. A three-car platoon and a roadside controller.

Followers have a *link* to their leader, a leader knows a list of followers, and there are links between roadside controllers and leaders. Links are used like pointers, that is, a follower can query (or even modify) the state of its leader. In addition, an inherent requirement on a model of car-platooning is that it provides for cars dynamically entering and leaving the highway, that is, there is no finite upper bound on the number of cars present at a certain point in time.

Thus the class of systems we consider is characterised by (i) dynamic creation and destruction of processes of different kinds, (ii) local state with finite-domain and other variables, and (iii) local and global links. The considered properties are general LTL formulae with outermost quantification over processes.

A well-established approach to the formal verification of safety ("two different cars never consider each other to be leader") or liveness ("a merge request is finally answered") properties of transition systems with large or infinite state-space are so-called finitary abstractions [2]. A finitary abstraction is defined by a finite abstract domain and a state abstraction mapping states (like the one shown in Figure 1(b)) to abstract representations of states. The set of initial abstract states and the transition relation are then induced by the state abstraction; abstract states are initial (or in transition relation) if they are the abstraction of initial states (or states in transition relation).

One of the oldest finitary abstractions is *Counter Abstraction* [3–5, 2]. The basic idea is to map states with many processes, each in one of finitely many local states, to an abstract state, which only counts how many processes are in each local state. Processes are considered equivalent if they share the same local state. To obtain a finite abstract domain, counters are typically cut off at two, i.e. distinguish only between 0, 1, and "many" processes. Such a state abstraction function α maps, for example, the concrete state on the left-hand side of Figure 2 to the one shown on the right where "many" processes are indicated by double-lines.

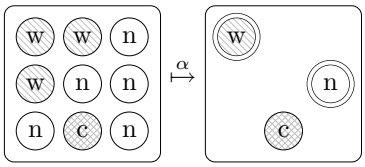


Fig. 2. Counter Abstraction.

Classical Counter Abstraction seems inappropriate to verify car platooning as it does not support links, only admits finite-domain variables, and suffers from the problem that processes *migrate* freely between equivalence classes such that we cannot tell whether *a particular* process has made a particular transition.

For example, if one process changes state from c to n and one from w to c in Figure 2, then the abstract state remains the same.

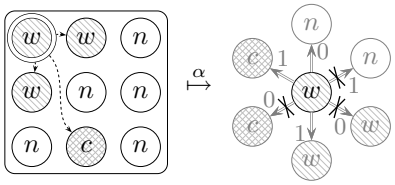


Fig. 3. Environment Abstraction.

Recently, Counter Abstraction was combined with a particular instance of Predicate Abstraction to a technique called *Environment Abstraction* (EA) [6], admitting infinite-domain local variables, like unbounded counters. EA derives its name from the way it addresses the migration problem: by representing one process precisely and preserving information about the rest from the perspective of this process in terms of binary so called inter-predicates on the unbounded variables. In Figure 3, dashed lines indicate whether the single inter-predicate holds between another and the reference process (indicated by double-line). An abstract state consists of the reference process' local state and a vector of bits indicating whether there is at least one other process in a particular local state and in inter-predicate relation to the reference process, i.e. counters are already cut off at 1. For example, in Figure 3 there are no processes in local state n and in inter-predicate relation to the reference process, thus in the abstract state the north-east arrow is crossed out. The north-arrow is not crossed out as there are n-processes *not* in inter-relation. *Not supporting links*, EA seems inappropriate for platooning as well. Furthermore, EA cannot verify manoeuvres involving more than two cars, because it is restricted to two-process safety and one-process liveness properties.

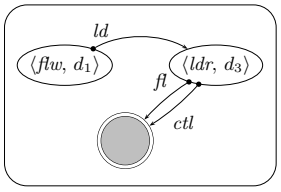


Fig. 4. Data-Type Red.

The finitary abstraction *Data-Type Reduction* (DTR) supports links and the desired properties, however, it does not admit infinite-domain variables like counters. DTR was introduced as part of a compositional verification methodology for parameterised systems [7]. The underlying idea is to represent the local state of finitely many processes exactly, like links between reference processes, remember whether there are links into their environment, and dismiss any other information about their environment. For example, the state in Figure 1(b) maps to the abstract state shown in Figure 4 if the leader and the last follower are reference processes. One gray summary node represents all other cars and all roadside controllers.

EA and DTR come close to a good abstraction technique for platooning. They have complementary strengths: DTR supporting links and manoeuvres with more than two cars, and EA supporting infinite-domain local state. They share the common idea of keeping some distinguished processes exact, intuitively putting a "spotlight" on them, while abstracting from the rest. Therefore, we aim for a combination of EA and DTR in order to treat systems like the car-platooning example. As they are formalised in rather different manners and have some undesired restrictions, we use the general and powerful framework

of canonical abstraction (CA) [8] to re-formulate in a common language and, ultimately, combine the concepts behind EA and DTR.

Canonical abstraction provides a general framework for concise and clear definition of state abstractions. It is widely used in the context of heap-manipulating programs. For example, the simplest instance of CA discussed in [8] maps a state with the linked list shown in Figure 5 to the abstract state at the bottom where nodes indistinguishable via links x and y collapse. Links into and between the summary become indefinite as, for example, some summarised nodes point to the last one and some do not (cf. [8] for details). Thus CA provides natural means to handle structures with links, and, as it turns out, for the principle to represent the environment from the perspective of reference individuals. We obtain an alternative elegant soundness proof of EA and DTR via the framework of CA. This has practical relevance since, in practice, abstractions often need to be refined in order to be precise enough. As a consequence of the CA framework and contrary to the original formalizations of EA and DTR, abstraction refinement becomes a natural process with guaranteed soundness.

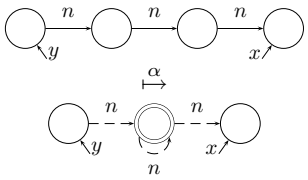


Fig. 5. Singly Linked List.

Other Abstractions and Related Work The static analysis-based approach of Yahav [9] first demonstrated suitability of the CA framework for the verification of concurrent Java programs with unbounded creation of processes on the heap. The idea underlying the employed abstraction is similar to EA and DTR, but the approach is limited to safety properties (or state invariants). This approach is refined in [10] by demonstrating that splitting a given task into cases and treating each case separately with a specially tailored abstraction gains efficiency, and that keeping neighbours of reference processes precise gains precision.

Yahav, Reps, Sagiv, and Wilhelm [11] address the same class of systems but use significantly stronger *Evolutional Temporal Logic* (ETL) properties, which are basically LTL with arbitrary quantification over processes (not only outermost) and transitive closure. Their approach is different to finitary abstraction in that they construct a set of abstract sequences of abstract states via static analysis. ETL formulae are then checked on this set of abstract traces, where consecutive similar states collapse to summary states and where evolution of processes is explicitly traced between (abstract) states of an abstract trace.

A thorough discussion why approaches, from instances of Predicate Abstraction to indexed predicates, are also insufficient appears in [6].

Most closely related to the aspect of our work, that we compare two independently developed and described state abstractions in the CA framework is Manevich et al. [12] who compare particular state abstractions for linked lists given via CA to equivalent Predicate Abstractions. Thus they compare single state abstractions in different frameworks.

Outline We proceed as follows. In Section 2 we formally define the class of systems and properties we consider. Section 3 introduces state abstractions with

respect to reference processes and briefly provides the Canonical Abstraction framework. In Section 4, we give native and CA-based definitions of DTR and EA and propose a combination in Section 5. Section 6 concludes.

2 Computational Model and Property Specification

In order to represent the car-platooning example from the introduction we consider transition systems over signatures. A signature $\mathcal{S}$ consists of process types T (like cars and roadside controllers), global links G and links local to processes L (like cars' link to the leader ld), and finite- and infinite-domain variables X and Y local to processes (like cars' current destination d and the controllers' highway utilisation y), all five sets disjoint, and a domain $\mathcal{D}$ assigning each variable $v \in X \cup Y$ a domain $\mathcal{D}(v)$, which is finite if $v \in X$. That is, $\mathcal{S} = (T, G, L, X, Y, \mathcal{D})$.

A transition system is a triple (S, S_0, R) of a set of states S, initial states $S_0 \subseteq S$, and a transition relation $R \subseteq S \times S$. It is called transition system over $\mathcal{S}$ iff each state $s \in S$ is a structure of $\mathcal{S}$, that is, a pair (U, σ) of a set of individuals U, called universe, which is partitioned into one (possibly empty) partition per type in T and σ is a valuation of G, L, X and Y, that is,

- global links $g \in G$ are assigned individuals, i.e. $\sigma|_G : G \to U$,
- local links $l \in L$ and variables $v \in X \cup Y$ are assigned functions mapping individuals to other individuals or values, i.e. $\sigma|_L : L \to (U \to U)$ and $\sigma|_{X \cup Y} : L \to (U \to \mathcal{D})$.

2.1 Parameterised Systems

DTR and EA originally address *parameterised systems*, that is, systems where $K \in \mathbb{N}$ processes execute a single program in parallel, so we also introduce a rather general notion of parameterised systems over signatures. As we do not aim at exploiting a particular description language, we do not specify one but consider $\mathcal{M}$ to be a finite behavioural description over a signature $\mathcal{S}$ with n process types which, given a tuple $(K_1, \ldots, K_n) \in \mathbb{N}^n$, determines a transition system over $\mathcal{S}$ whose state-set consists of all structures (U, σ) of $\mathcal{S}$ over a fixed universe U with K_i individuals of type τ_i, $1 \leq i \leq n$.

The set of all such instances of $\mathcal{M}$ is called $\mathcal{M}(\mathbb{N})$. Note that each instance has only finitely many processes, the challenge of parameterised system verification is to verify all instances at once.

In addition to common practice we use $\mathcal{M}(\infty)$ to denote the set of instances with countably infinitely many processes of some type because systems with a dynamically changing number of processes, like car platooning, can be encoded therein [13]. For the Canonical Abstraction versions of DTR and EA in Section 4, it is more suitable to consider the single transition system obtained by taking the union of all instances, denoted by $\mathcal{M}^{\mathbb{N}}$ etc., instead of a set of transition systems.

$\mathit{type}[\tau]/1$	the given individual is of type $\tau \in T$
$\mathit{ref}[g]/1$	the global link $g \in G$ points to the given individual
$\mathit{val}[x, d]/1$	the local variable $x \in X$ has value $d \in \mathcal{D}$
$\mathit{ref}[l]/2$	the local link $l \in L$ of the given individual points to the other one
$\mathit{eq}/2$	the two given individuals are equal

Table 1. Signature Predicates. Symbol p being of arity k is indicated by p/k.

2.2 Properties

As Canonical Abstraction operates on logical structures (cf. Section 3.2), it is useful to only consider properties in form of formulae over a finite set of predicate symbols $\mathcal{P}$. Given a signature $\mathcal{S}$, we consider the set $\mathcal{P}_\mathcal{S}$ consisting of the predicate symbols given by Table 1. Note that the predicate symbols in $\mathcal{P}_\mathcal{S}$ distinguish the complete information about links and processes' finite variables in a state, thus together with quantification over processes we do not lose generality on these aspects by considering only $\mathcal{P}_\mathcal{S}$.

For EA, we in addition need a set of binary predicate symbols that typically relate the non-finite aspects of two processes. In the car-platooning example it could compare the real-valued utilisation of two roadside controllers. We assume that a parameterised system $\mathcal{M}$ defines a finite set $\mathcal{P}_\mathcal{M}$ of these *inter-predicates*.

A structure $s = (U, \sigma)$ induces an interpretation ι_s of the predicate symbols in $\mathcal{P}_\mathcal{S}$. For example, $\iota_s(\mathit{val}[x, d])$ holds for $u \in U$ iff $\sigma(u)(x) = d$. For each inter-predicate $p \in \mathcal{P}_\mathcal{M}$ we assume an interpretation $\iota_s(p) : U^2 \to \{0, 1\}$ to be given. In general, a pair (U, ι) of a universe U and an interpretation ι of a set of predicate symbols $\mathcal{P}$ is called two-valued logical structure of $\mathcal{P}$. The set of all two-valued logical structures of $\mathcal{P}$ is denoted by $\mathit{2\text{-}Struct}[\mathcal{P}]$.

The language of *evolution properties* consists of formulae of the form

$$\forall\, z_1, \ldots z_n \,.\, \phi,\ n \in \mathbb{N}_0 \tag{1}$$

where $z_1, \ldots, z_n$ are logical variables (without loss of generality denoting *different* processes) and ϕ is an LTL formula with $\mathbf{X}$ (Next), $\mathbf{U}$ (Until), $\mathbf{G}$ (Globally), and $\mathbf{F}$ (Finally) and logical connectives over non-temporal state invariants

$$\psi \in \mathit{SF} ::= z_1 = z_2 \mid p(z_1, \ldots, z_n) \mid \neg\psi_1 \mid \psi_1 \vee \psi_2 \mid \exists\, z_1 \,.\, \psi_1 \tag{2}$$

where p is a predicate symbol from $\mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{M}$. DTR supports all *evolution properties*, while EA is restricted to properties of the forms

$$\forall\, z_1, z_2 \,.\, \mathbf{G}\ \psi(z_1, z_2) \quad \text{and} \quad \forall\, z_1 \,.\, \mathbf{G}\ (\psi_1(z_1) \to \mathbf{F}\ \psi_2(z_1)) \tag{3}$$

over the predicate symbols $\mathit{val}[x, d]$. The former are called called *two-indexed safety properties*, the latter *one-indexed properties*. The semantics of a state invariant ψ in a state s, denoted by $[\![\psi]\!]_s$, and satisfaction of an evolution property by a sequence of states $\pi = (U_n, \sigma_n)_{n \in \mathbb{N}}$, denoted $\pi \models \Phi$, is inductively defined based on the logical structure $(U, \iota_s) \in \mathit{2\text{-}Struct}[\mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{M}]$ induced by state s.

2.3 Augmentation

As outlined in the introduction, both, DTR and EA, depend on a set of designated reference processes. To provide both uniformly with reference processes, we'll employ a simple, technical procedure that has similarly been applied, e.g. by [2] in the context of safety and liveness properties of parameterised systems and by [14] in the context of shape analysis for list insertion.

Given a transition system M over a signature $\mathcal{S}$ with global links G, let $G_a = \{g_{a_1}, \ldots, g_{a_n}\}$ be a set of fresh global *augmentation links*. Then the G_a-*augmentation* of M is a transition system $\widehat{M}$ over $\widehat{\mathcal{S}}$ with global links $G \cup G_a$ where the augmentation links consistently trace n different individuals. Consistency means that the valuation of G_a is constant over transitions, i.e.

$$((U, \widehat{\sigma}_1), (U, \widehat{\sigma}_2)) \in \widehat{R} \implies \widehat{\sigma}_1|_{G_a} = \widehat{\sigma}_1|_{G_a}. \tag{4}$$

States of $\widehat{M}$ are initial (in transition relation), if the projection onto $\mathcal{S}$ is initial (in transition relation). Among others, Figure 6 illustrates augmentation.

Then, for example, a formula $\forall\, z_1, z_2 \,.\, \mathbf{G}\; p(z_1, z_2)$ holds in M iff

$$\mathbf{G}\; (\forall\, z_1, z_2 \,.\, (\mathit{ref}[g_{a_1}](z_1) \wedge \mathit{ref}[g_{a_2}](z_2)) \rightarrow p(z_1, z_2)) \tag{5}$$

holds in $\widehat{M}$, the $\{g_{a_1}, g_{a_2}\}$-augmentation of M. The example easily extends into an inductive definition of the transformation of evolution properties Φ into $\widehat{\Phi}$.

3 Defining and Comparing State Abstractions

A state abstraction of a transition system $M = (S, S_0, R)$ consists of an abstract domain $S^\sharp$ and a state abstraction function α mapping concrete to abstract states, i.e. $\alpha : S \to S^\sharp$. It is called finite if $S^\sharp$ is finite. The function α induces an abstract transition system M_α with state-set $S^\sharp$ by considering an abstract state initial iff it is the abstraction of an initial concrete state, and two abstract states in transition relation if they are the abstractions of two concrete states in transition relation. This construction is known as *finitary abstraction* [15]. Together with α, we always consider its concretisation function γ mapping abstract states to the concrete states they represent, i.e. $\gamma(s^\sharp) = \{s \in S \mid \alpha(s) = s^\sharp\}$.

In order to establish properties of the original system on the abstract one, a state abstraction is complemented by a conservative, three-valued interpretation of the predicate symbols from $\mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{M}$ in each abstract state. An interpretation is called three-valued iff predicates map to $\{0, 1, 1/2\}$ instead of $\{0, 1\}$; by *3-Struct*$[\mathcal{P}]$ we denote the set of all pairs (U, ι) of universes and three-valued interpretations of the predicate symbols in $\mathcal{P}$.

An interpretation of predicate symbols $\mathcal{P}$ is called conservative with respect to another iff it doesn't introduce contradictions; in our case this spells out as

$$\forall\, p \in \mathcal{P}\; \forall\, s^\sharp \in S^\sharp\; \forall\, s \in \gamma(s^\sharp) : [\![p]\!]_s \sqsubseteq [\![p]\!]^\sharp_{s^\sharp} \tag{6}$$

where "$\sqsubseteq$" is the information order on $\{0, 1, 1/2\}$, defined as $\{b \sqsubseteq b, b \sqsubseteq 1/2 \mid b \in \{0, 1, 1/2\}\}$. Thus the third truth-value $1/2$ can be read as "don't know". Using the well-established three-valued semantics of state formulae [8] and temporal formulae [9], a conservative abstract semantics for temporal formulae is obtained. Thus if a property Φ holds in M_α, then it also holds in M.

3.1 Comparing State Abstractions

Recall that our overall aim is to provide alternative definitions of EA and DTR in the framework of Canonical Abstraction. In order to prove that the new definition is equivalent to the original one, we first introduce notions of equivalence and being coarser for state abstractions. The following Lemma provides more easily checkable, sufficient criteria that imply equivalence or being coarser.

A state abstractions $\alpha_1 : S \to S_1^\sharp$ is called coarser than $\alpha_2 : S \to S_2^\sharp$, denoted by $\alpha_1 \succeq \alpha_2$, iff the induced abstract transition system satisfies fewer evolution formulae, i.e. iff

$$M_{\alpha_1} \models \phi \implies M_{\alpha_2} \models \phi \tag{7}$$

for all evolution formulae Φ. Both are called equivalent, denoted $\alpha_1 \equiv \alpha_2$, iff $\alpha_2 \succeq \alpha_1$ and $\alpha_1 \succeq \alpha_2$, that is, if both satisfy the same properties.

If there is a simulation relation between the induced abstract models, (7) and thus the coarser-than relation follow. For existence of a simulation relation it is sufficient to find a relation ϱ between the two abstract domains such that related states do not interpret predicates contradictingly[3] and the states of the coarser state abstraction concretise to more concrete states.

Lemma 1 (State Abstraction Comparison). *Let $\alpha_1 : S \to S_1^\sharp$ and $\alpha_2 : S \to S_2^\sharp$ be two state abstractions. Let $\varrho : S_1^\sharp \times S_2^\sharp$ be a relation such that*

1. $\forall\, s \in S :\ (\alpha_1(s), \alpha_2(s)) \in \varrho$
2. $\forall\, (s_1^\sharp, s_2^\sharp) \in \varrho\ \forall\, p \in \mathcal{P} : [\![p]\!]^\sharp_{s_2^\sharp} \sqsubseteq [\![p]\!]^\sharp_{s_1^\sharp}$
3. $\forall\, (s_1^\sharp, s_2^\sharp) \in \varrho : \gamma_1(s_1^\sharp) \subseteq \gamma_2(s_2^\sharp)$

Then $\alpha_1 \succeq \alpha_2$. With "$=$" instead of "$\sqsubseteq$" and "$\subseteq$", $\alpha_1 \equiv \alpha_2$ is obtained.

3.2 Canonical Abstraction

Canonical Abstraction provides a framework for the definition of state abstraction functions if concrete states are three-valued structures of a finite set of predicate symbols $\mathcal{P}$. Following the framework, a choice of a set of unary, so-called abstraction predicates

$$\mathcal{A} = \{p_{a_1}, \ldots, p_{a_n}\} \subseteq \mathcal{P} \tag{8}$$

[3] assuming that the interpretation of formulae is inductively defined as discussed in the previous paragraph

determines the abstract domain as the set of three-valued structures (U, ι) where U comprises only the canonical names with respect to $\mathcal{A}$, thus it is finite.

The canonical name $\kappa_{\mathcal{A}}(u)$ of an individual u is simply the valuation of the abstraction predicates on u, i.e., the vector $(p_{a_1}(u), \ldots, p_{a_n}(u))$. The abstract domain is finite as there are only finitely many such vectors.

The other predicates from $\mathcal{P}$, which are not used as abstraction predicates, are principally only required to evaluate conservatively in the abstract state. A best abstraction with respect to $\mathcal{A}$ evaluates them as precisely as possible, that is, to a definite value from $\{0, 1\}$ if all summarised individuals agree on the definite value and to $1/2$ only otherwise.

The state abstraction function $\alpha_{\mathcal{A}} : \textit{3-Struct} \to \textit{3-Struct}$ is such a best abstraction. That is, as it merges individuals indistinguishable by the abstraction predicates, it preserves information about the abstraction predicates precisely, all other information may be blurred to $1/2$.

If defined by $\mathcal{A} \subseteq \mathcal{P}_{\mathcal{S}} \cup \mathcal{P}_{\mathcal{M}}$, it naturally extends to states that are pairs $s = (U, \sigma)$ of a universe and a valuation of signature $\mathcal{S}$ by applying it to the induced structure (U, ι_s), that is, by setting $\alpha_{\mathcal{A}}(s) := \alpha_{\mathcal{A}}(U, \iota_s)$. The concretisation function is still defined as on page 7.

Formally, Canonical Abstraction is based on the notion of (tight) embedding of three-valued structures. A surjection $h : U \to U'$ between two universes is said to embed the logical structure $s = (U, \iota)$ of $\mathcal{P}$ into $s' = (U', \iota')$ iff

$$\forall\, p \in \mathcal{P}^k : \iota(p)(u_1, \ldots, u_{k(p)}) \sqsubseteq \iota'(p)(h(u_1), \ldots, h(u_{k(p)})). \tag{9}$$

The embedding is called tight, if the stronger condition

$$\forall\, p \in \mathcal{P}^k : \iota'(p)(u'_1, \ldots, u'_{k(p)}) = \bigsqcup_{h(u_i)=u'_i, 1 \le i \le k(p)} \iota(p)(u_1, \ldots, u_{k(p)}), \tag{10}$$

using the least upper bound with respect to information order, holds. A structure s *can* (tightly) be embedded into s' iff a (tight) embedding function exists.

Given the three-valued interpretation $[\![\psi]\!]^3_{s'}$ of state invariants in abstract states s' via the monotone Kleene semantics[4] the following theorem holds

Theorem 1 (Embedding Theorem [8]). *Let $s = (U, \iota)$ and $s' = (U', \iota')$ be logical structures, let h embed s in s', and let Z be a complete assignment of the free variables in ψ. Then $[\![\psi]\!]^3_s(Z) \sqsubseteq [\![\psi]\!]^3_{s'}(h \circ Z)$.*

4 The Spotlight Principle

Intuitively, both EA and DTR focus, or put a spotlight, on one or more processes and abstract from the rest, the ones in the shadows. Information about the latter is kept from the perspective of the spotlight individuals.

[4] comparison of the summary node with itself then yields $1/2$ if there is more than one individual represented by a summary node, which is always the case in Section 4

We say that a state abstraction α *follows the spotlight principle* if it is definable via Canonical Abstraction and there are abstraction predicates p_a in $\mathcal{A}$ that concretise to at most one individual in each abstract state, i.e.

$$\forall\,(U,\iota) \in S^\sharp : |\{u \mid \iota(p_a)(u)\}| \leq 1. \tag{11}$$

A direct consequence is that all other unary predicates in $\mathcal{P}_\mathcal{S}$ are evaluated to definite values for a spotlight individual (or reference process); binary predicates may evaluate to $1/2$ if evaluated for non-spotlight individuals.

We call α disjoint, if spotlight predicates $p_{a_1}, \ldots, p_{a_n}$ mutually exclude each other on individuals. Given a transition system M over a signature, an evolution formula $\Phi = \forall\, z_1, \ldots, z_n \,.\, \phi$, and a corresponding G_a-augmentation $\widehat{M}$ of M with $G_a = \{g_{a_1}, \ldots, g_{a_n}\}$, each state abstraction

$$\mathcal{A} \supseteq \{\mathit{ref}[g_a] \mid g_a \in G_a\} \tag{12}$$

is a disjoint spotlight abstraction.

In the following, we present the two abstractions EA and DTR in their original definition and give an equivalent Canonical Abstraction definition for each. Thereby, both can be identified as successful applications of the spotlight principle. In Section 5, we can then use the insights gained in the following sections to combine both abstractions into one which allows to treat the example from the introduction, which is neither in the scope of DTR nor in the scope of EA. For completeness, we additionally compare both to a typical example of the abstractions that are usually given via Canonical Abstraction, namely Shape Analysis of programs manipulating linked lists.

4.1 Data-Type Reduction

Data-Type Reduction [7] (DTR) has been introduced for parameterised systems over signatures without infinite domain variables, i.e. $Y = \emptyset$, thus the considered systems are only infinite by the number of instantiations in $\mathcal{M}(\mathbb{N})$, or the number of processes in $\mathcal{M}(\infty)$.

In the following, let $\mathcal{M}$ be a parameterised system over signature $\mathcal{S}$ with $Y = \emptyset$ and, as DTR depends on the property, let $\Phi = \forall\, z_1, \ldots, z_n \,.\, \phi(z_1, \ldots, z_n)$ be an evolution property. Let $M \in \mathcal{M}(\infty)$ be the transition system with infinitely many processes of each type and $\widehat{M}$ a G_a-augmentation corresponding to Φ.

Native Definition The finite state abstraction function $\alpha_{dtr} : S \to S^\sharp$ maps states $(U, \sigma) \in S$ to abstract states $(U^\sharp, \sigma^\sharp)$ where $\sigma^\sharp$ maps global links from G_a to the corresponding abstract individuals, i.e.

$$\sigma^\sharp(g_{a_i}) = u_i^\sharp,\ g_{a_i} \in G_a, \tag{13}$$

and local and other global links, $g \notin G_a$, to the corresponding abstract individual or the summary individual $u_0^\sharp$, i.e.

$$\sigma^\sharp(g) = \begin{cases} u_i^\sharp & , \sigma(g_{a_i}) = \sigma(g) \\ u_0^\sharp & , \text{otherwise} \end{cases} \quad \sigma^\sharp(l)(u_i^\sharp) = \begin{cases} u_j^\sharp & , \sigma(l)(\sigma(g_{a_i})) = \sigma(g_{a_j}) \\ u_0^\sharp & , \text{otherwise} \end{cases} \tag{14}$$

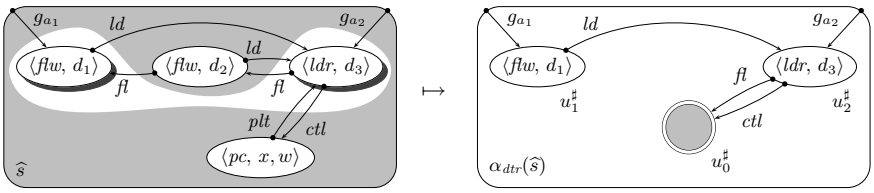
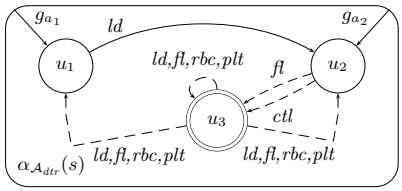


Fig. 6. Data-Type Reduction.



	g_{a_1}	g_{a_2}	*car*	*ctl*	*flw*	...
u_1	1	0	1	0	1	...
u_2	0	1	1	0	0	...
u_3	0	0	1/2	1/2	1/2	1/2

fl	u_1	u_2	u_3
	0	0	0
	0	1/2	0
	1/2	1/2	1/2

Fig. 7. DTR via Canonical Abstraction. The tables exemplary show the valuation of some predicates, the unary reference individual predicates, the type predicates, and $val[st, flw]$ on the left and the binary predicate $ref[fl]$ on the right.

and keeps the values of local variables, i.e. $\sigma^\sharp(x)(u_i^\sharp) = \sigma(x)(\sigma(g_{a_i}))$.

Figure 6 illustrates the effect of α_{dtr} on a state of the car platooning system from Section 2 (assuming $w \in X$, instead of $y \in Y$). The abstract state preserves the state of the last follower and the leader. Links to individuals in the shadows change to links to the summary individual, links from them are lost.

The interpretation of a predicate $p \in \mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{M}$ of arity k in $s^\sharp$ is defined as

$$[\![p]\!]^\sharp_{s^\sharp}(w_1^\sharp, \ldots, w_k^\sharp) = 1/2 \tag{15}$$

if one of the individuals is the summary individual, i.e. $w_i^\sharp = u_0^\sharp$ for some $1 \leq i \leq n$, and the value obtained using the regular definition from Section 2.2 otherwise. We immediately have $[\![p]\!]_s(u_1, \ldots, u_n) \sqsubseteq [\![p]\!]^\sharp_{\alpha s}(w_1^\sharp, \ldots, w_n^\sharp)$ if u_i and $w_i^\sharp$ are indistinguishable on the reference link predicates $ref[g_a]$, $g_a \in G_a$.

Data-Type Abstraction via Canonical Abstraction is obtained by choosing the reference individual predicates as abstraction predicates, i.e.

$$\mathcal{A}_{dtr} = \{ref[g] \mid g_a \in G\} \subseteq \mathcal{P}_\mathcal{S}. \tag{16}$$

Figure 7 illustrates, following the conventions of [8], the effect of $\alpha_{\mathcal{A}_{dtr}}$ on the concrete state from Figure 6. Dashed (indefinite) edges indicate the loss of precision that shows in the original definition only in the evaluation of expressions.

Note that $\alpha_{\mathcal{A}_{dtr}}$ is already too precise as it preserves information about the shadow individuals if predicates happen to agree on all of them. An equivalent state abstraction can be obtained by explicitly blurring the truth-value of all predicates, except for the spotlight predicates $ref[g_a]$, when evaluated for at least

one non-reference individual, i.e. we set $\alpha'_{\mathcal{A}_{dtr}} := blur \circ \alpha_{\mathcal{A}_{dtr}}$ where $blur(U, \iota) := (U, blur(\iota))$ with

$$blur(\iota)(p)(u_1, \ldots, u_n) = \begin{cases} 1/2 & \text{, if } p \neq ref[g_a], g_a \in G_a, \text{ and} \\ & \bigwedge_{\substack{g_a \in G_a \\ 1 \leq i \leq n}} \neg \iota(ref[g_a])(u_i) \\ \iota(p)(u_1, \ldots, u_n) & \text{, otherwise} \end{cases} \tag{17}$$

Theorem 2. *The native definition of DTR α_{dtr} is equivalent to $\alpha'_{\mathcal{A}_{dtr}}$.*

Proof. By Lemma 1, letting abstract DTR-states $s^\sharp_{dtr}$ and $s^\sharp_{\mathcal{A}_{dtr}}$ be ϱ-related iff

$$\forall\, p \in \mathcal{P}_\mathcal{S} \cup \mathcal{P}_\mathcal{M} : [\![p]\!]^\sharp_{s^\sharp_{\mathcal{A}_{dtr}}}(u_1, \ldots, u_n) = [\![p]\!]^\sharp_{s^\sharp_{dtr}}(u^\sharp_1, \ldots, u^\sharp_n) \tag{18}$$

for u_i and $u^\sharp_i$ indistinguishable under $ref[g_a]$, $g_a \in G_a$. □

4.2 Environment Abstraction

Environment Abstraction [6] (EA) has been introduced for parameterised systems over signatures with exactly one process type and no links. Thus the considered systems are infinite by the number of instantiations $\mathcal{M}(\mathbb{N})$, or the number of processes in $\mathcal{M}(\infty)$, and in addition possibly by the domain of variables in Y.

In the following, let $\mathcal{M}$ be a parameterised system over signature $\mathcal{S}$ without links and one process type. to simplify the presentation, we follow [6] in assuming that X comprises only the single, finite-domain variable *pc*.

The car-platooning example from the introduction is clearly out of scope for EA as it depends on links between processes. So we employ one (of the two) examples that have successfully been verified with EA [6], namely the parameterised system employing the bakery algorithm [16] for mutual exclusion. Assume, the program counter *pc* has a domain of three locations like n (non-critical), w (wait), c (critical) and there is one (unbounded) integer variable t for the ticket.

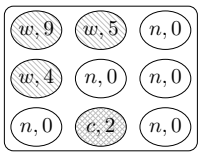


Fig. 8. Bakery State.

Figure 8 shows one state of bakery with $K = 9$ processes. Oval nodes represent processes, giving their state (also indicated by different hatch fillings) and ticket value, assuming idle processes reset the ticket to 0.

In the following, let $M \in \mathcal{M}(\infty)$ be the transition system with infinitely many processes (of the only type) and $\widehat{M}$ an augmentation with a single link g_a.

Native Definition In [6], a set of predicates $env[i, j]$ is constructed in two steps. Let $\mathcal{P}_\mathcal{M} = \{p_1, \ldots, p_n\}$ be the inter-predicates of $\mathcal{M}$. Then firstly there are 2^n formulae R_i with two free variables characterise all (mutually exclusive) combinations of the inter-predicates holding or not for two individuals, i.e.

$$R_i(z_1, z_2) := \pm p_1(z_1, z_2) \wedge \cdots \wedge \pm p_n(z_1, z_2) \tag{19}$$

The R_i secondly induce $T := 2^n \cdot |\mathcal{D}(pc)|$ so-called *environment formulae* holding in state (U, σ) if at least one individual different from the reference individual has pc value j and is related to the reference individual as described by R_i, i.e.

$$env[i, j] := \exists\, z, z' \,.\, z \neq z' \wedge ref[g_a](z) \wedge R_i(z, z') \wedge val[pc, j](z') \tag{20}$$

The abstract domain $S^\sharp$ of the EA of $\mathcal{M}$ is the set of vectors

$$\langle d, \epsilon_{1,1}, \ldots, \epsilon_{2^n, |\mathcal{D}(pc)|} \rangle \in \mathcal{D}(pc) \times \{0, 1\}^T \tag{21}$$

comprising a pc-value $d \in \mathcal{D}(pc)$ and one boolean $\epsilon_{i,j}$ for each of the T environment formulae $env[i, j]$. It is finite as $\mathcal{D}(pc)$ is finite.

The finite state abstraction function $\alpha_{ea} : \widehat{S} \to S^\sharp$ maps states $\widehat{s} = (U, \sigma) \in \widehat{S}$ to the vector $\langle \sigma(pc)(\sigma(g_a)), [\![env[1, 1]]\!]_{\widehat{s}}, \ldots, [\![env[2^n, |\mathcal{D}(pc)|]]\!]_{\widehat{s}} \rangle$.

Figure 3 illustrates the effect of α_{ea} on an augmented state. Note that the valuation of inter-predicates is only shown with respect to the reference individual. The abstraction function α_{ea} keeps the value of pc for the reference process and one bit for each combination of program counter and inter-predicate being 0 iff there is no other process with a corresponding pc in the concrete state such that the inter-predicate holds. In other words, each $\epsilon_{i,j}$ encodes presence or absence of at least one individual that is in $env[i, j]$ relation to the reference individual.

The interpretation of a unary predicate $val[pc, d] \in \mathcal{P}_\mathcal{S}$ is defined using a structure (U, ι) with an arbitrary, two-individual universe $\{u_1, u_2\}$ and

$$\iota(val[pc, d]) = \{u_1 \mapsto (pc = d), u_2 \mapsto \bigvee_{1 \leq i \leq 2^n} \epsilon_{i,d}\} \tag{22}$$

for an abstract state $s^\sharp = \langle pc, \epsilon_{1,1}, \ldots, \epsilon_{2^n, |\mathcal{D}pc|} \rangle$. Then $[\![val[pc, d]]\!]^\sharp_{s^\sharp} = \iota(val[pc, d])$. Intuitively, $val[pc, d]$ holds in $s^\sharp$ if either the first component of the vector is equal to d or at least one $\epsilon_{i,d}$, $1 \leq i \leq 2^n$, is true.

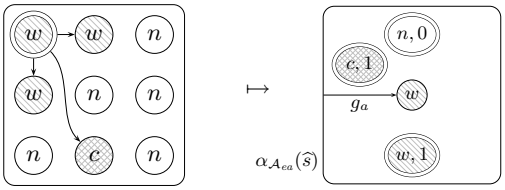
Environment Abstraction via Canonical Abstraction is based on a slightly different set of environment predicates. Let $env[p]$, $p \in \mathcal{P}_\mathcal{M}$ be unary predicate symbols that indicate whether an individual is *not* the reference individual and in p-relation to the reference individual, i.e.

$$[\![env[p]]\!]_{(U,\sigma)}(u) := (u \neq \sigma(g_a) \wedge [\![p]\!]_{(U,\sigma)}(\sigma(g_a), u)) \tag{23}$$

Then as abstraction predicates we choose the one for the reference individual, for finite-domain variable valuation, and the new environment predicates, i.e.

$$\mathcal{A}_{ea} = \{ref[g_a]\} \cup \{val[pc, d] \mid d \in \mathcal{D}pc\} \cup \{env[p] \mid p \in \mathcal{P}_\mathcal{M}\} \tag{24}$$

Figure 9 illustrates the effect of CA with $\mathcal{A}_{ea}$ on the concrete state from Figure 3. Note that there are no edges between nodes as we do not have binary predicates in $\mathcal{P}$ and as all predicates in $\mathcal{P}$ are abstraction predicates. Loss of



	g_a	n	w	c	$env[p_1]$	sm
u_1	1	0	1	0	0	0
u_2	0	0	1	0	1	1/2
u_3	0	1	0	0	0	1/2
u_4	0	0	0	1	1	1/2

Fig. 9. EA via Canonical Abstraction. The table shows the valuation of all predicates considered in the Bakery example and the summary predicate *sm*.

precision takes place in the choice of predicates which, in contrast to DTR, doesn't preserve all information of concrete states.

Similar to DTR, the more natural choice of abstraction predicates, namely

$$\mathcal{A}'_{ea} = \{ref[g_a]\} \cup \{val[pc, d] \mid d \in \mathcal{D}pc\} \tag{25}$$

is already more precise than the original definition of EA as it would preserve information on the relation *between* the individuals in the shadows.

Theorem 3. *The native definition of EA α_{ea} is equivalent to $\alpha_{\mathcal{A}_{dtr}}$.*

Proof. By Lemma 1, letting abstract states $s^\sharp_{ea} = \langle d, \epsilon_{1,1}, \ldots, \epsilon_{2^n,|\mathcal{D}(pc)|}\rangle$ and $s^\sharp_{\mathcal{A}_{ea}} = (U, \iota)$ be ϱ-related iff $\epsilon_{i,j} = [\![\exists\, u'.\ p_i(u, u') \wedge val[pc, j](u) \wedge ref[g](u)]\!]_{s^\sharp}$ and $\bigvee_{j\in\mathcal{D}}(d = j \Leftrightarrow [\![\exists\, u.\ val[pc, j](u) \wedge val[pc, j](u)]\!])$. □

4.3 Shape Analysis

A natural question is how EA and DTR relate to the abstractions for which Canonical Abstraction is typically used (cf. Section 1). The abstraction predicates of the coarsest abstraction for linked lists in [8] are $\mathcal{A} = \{ref[x], ref[y]\}$. As program variables refer to at most one individual at a time, the abstractions for singly linked lists also follow the spotlight principle (although not disjointly).

This observation doesn't contradict the intuition that program variables *change* on update, while augmentation is constant. The abstraction used for linked lists is on such a high level of abstraction that it concretises as well to topologies of interlinked processes where x denotes a fixed process; the expectation that the value of x necessarily changes, exists only in the eye of the beholder.

5 Combining DTR and EA

As discussed in the introduction, both DTR and EA alone are not sufficient to establish properties like liveness of the merge procedure of car-platooning as DTR excludes infinite-domain variables and EA doesn't handle links between cars and is restricted to properties over at most two processes.

Furthermore, DTR doesn't preserve invariants about individuals outside the spotlight. In practice, this tends to give rise to spurious counter-examples, which have to be excluded by user-supplied non-interference lemmata [7, 13].

Given the formulation of both, DTR and EA, in the Canonical Abstraction framework a sound abstraction that combines the strengths of both is obtained by simply taking the union of their abstraction predicates, i.e. $\mathcal{A} := \mathcal{A}_{dtr} \cup \mathcal{A}_{ea}$. As adding abstraction predicates makes abstractions more precise, the state abstraction defined by $\mathcal{A}$ is more precise than both, DTR and EA. From EA it inherits support of unbounded local state variables and from DTR support for links and multiple process types in general evolution logic formulae.

Practically, stating a state abstraction is only one aspect, the other one is finding an implementation, which computes the abstract finite-state transition system directly without the need to explicitly enumerate the concrete, infinite state space. Specialised implementations for DTR and EA proposed in [7] and [6]. In contrast, the Canonical Abstraction framework is *generally* supported by tools like *TVLA* [17] and *bohne* [18] for the verification of state invariants. Due to their generality, a non-optimised application of, e.g., TVLA to DTR or EA may not be as efficient as the procedures of [7, 6], but they provide for easy prototyping when refining abstractions. One of the authors successfully implemented the variant of DTR given in Section 4.1 in TVLA to verify mutual exclusion for the bakery algorithm [19]. There the unbounded counter domain is modeled and abstracted by the list-like abstraction described in [9] admitting only increment and decrement operations. The ability of CA to preserve information about individuals in the shadow proved crucial to verify mutual exclusion.

6 Conclusion

There is a need for state abstractions suitable to treat systems with dynamic links between processes and infinite-domain variables and general temporal properties. From the literature, DTR and EA come closest but neither one is sufficient.

In order to obtain a combination with the strengths of both, we stated them uniformly in the Canonical Abstraction framework, which is a new application of the framework. By comparison of the employed abstraction predicates it turns out that both DTR and EA share a common principle which we call the spotlight principle. Individuals in the spotlight are kept precise while information about the others is represented from the perspective of those in the spotlight.

Stating other abstractions like [9, 10] in this framework in order to dissect the ideas employed there remains for the full version of the paper. Further work comprises an investigation of the effect of cutting off counters at 2, as it is done for Shape Analysis, instead of at 1. Another question concerns the other direction, i.e. whether particular abstractions stated via Canonical Abstraction may profit from the efficient implementations of DTR or EA. And we would like to gain a deeper insight into the consequences of the spotlight principle, i.e. whether a set of preserved properties (possibly along segments of computation paths) can be characterised.

Acknowledgements. The authors want to express their gratitude to Andreas Podelski and Reinhard Wilhelm for their valuable comments on early versions of this work.

References

1. Hsu, A., Eskafi, F., Sachs, S., Varaiya, P.: The design of platoon maneuver protocols for IVHS. PATH Research Report UCB-ITS-PRR-91-6, Institute of Transportation Studies, University of California at Berkeley (1991) ISSN 1055-1425.
2. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,infty)-counter abstraction. In Hunt, Jr., W.A., Somenzi, F., eds.: Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. Volume 2725 of Lecture Notes in Computer Science., Springer (2003) 107–133
3. Lubachevsky, B.D.: An approach to automating the verification of compact parallel coordination programs. Acta Informatica **21** (1984) 125–169
4. Pong, F., Dubois, M.: Formal verification of complex coherence protocols using symbolic state models. J. ACM **45** (1998) 557–587
5. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39** (1992) 675–735
6. Clarke, E.M., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In Emerson, E.A., Namjoshi, K.S., eds.: Verification, Model Checking, and Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings. Volume 3855 of Lecture Notes in Computer Science., Springer (2006) 126–141
7. McMillan, K.L.: Verification of infinite state systems by compositional model checking (charme). In Pierre, L., Kropf, T., eds.: Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings. Volume 1703 of Lecture Notes in Computer Science., Springer (1999) 219–234
8. Sagiv, S., Reps, T.W., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ACM Transactions on Programming Languages and Systems **22** (2001)
9. Yahav, E.: Verifying safety properties of concurrent Java programs using 3-valued logic. ACM SIGPLAN Notices **36** (2001) 27–40
10. Yahav, E., Ramalingam, G.: Verifying safety properties using separation and heterogeneous abstractions. In: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation, ACM Press (2004) 25–34
11. Yahav, E., Reps, T., Sagiv, S., Wilhelm, R.: Verifying temporal heap properties specified via evolution logic. In Degano, P., ed.: Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. Number 2618 in Lecture Notes in Computer Science, Springer-Verlag (2003) 204–222
12. Manevich, R., Yahav, E., Ramalingam, G., Sagiv, M.: Predicate abstraction and canonical abstraction for singly-linked lists. In Cousot, R., ed.: Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings. Volume 3385 of Lecture Notes in Computer Science., Springer (2005) 181–198
13. Damm, W., Westphal, B.: Live and Let Die: LSC-based Verification of UML-Models. In Boer, F., Bonsangue, M., Graf, S., de Roever, W.P., eds.: Formal Methods for Components and Objects First International Symposium, FMCO

2002, Leiden, The Netherlands, November 5-8, 2002, Revised Lectures. Number 2852 in Lecture Notes in Computer Science, Springer-Verlag (2003) 99–135
14. Dams, D., Namjoshi, K.S.: Shape analysis through predicate abstraction and model checking. In: VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation, London, UK, Springer-Verlag (2003) 310–324
15. Kesten, Y., Pnueli, A.: Control and data abstraction: The cornerstones of practical formal verification. International Journal on Software Tools for Technology Transfer **2** (2000) 328–342
16. Lamport, L.: A new solution of dijkstras concurrent programming problem. Communications of the ACM **17** (1974) 453–455
17. Lev-Ami, T., Sagiv, M.: Tvla: A system for implementing static analysis. In Palsberg, J., ed.: Static Analysis, 7th International Symposium, SAS 2000, Santa Barbara, CA, USA, June 29 - July 1, 2000, Proceedings. Number 1824 in Lecture Notes in Computer Science, Springer-Verlag (2000) 280–301
18. Podelski, A., Wies, T.: Boolean heaps. In Hankin, C., Siveroni, I., eds.: Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings. Volume 3672 of Lecture Notes in Computer Science., Springer (2005) 268–283
19. Wachter, B.: Checking universally quantified temporal properties with three-valued analysis. Master's thesis, Universität des Saarlandes (2005)