Locality-Based Abstractions

Javier Esparza[1], Pierre Ganty[2,⋆], and Stefan Schwoon[1]

[1] Institut für Formale Methoden der Informatik, Universität Stuttgart
{esparza, schwoosn}@informatik.uni-stuttgart.de
[2] Département d'Informatique, Université Libre de Bruxelles
pganty@ulb.ac.be

Abstract. We present locality-based abstractions, in which a set of states of a distributed system is abstracted to the collection of views that some observers have of the states. Special cases of locality-abstractions have been used in different contexts (planning, analysis of concurrent programs, concurrency theory). In this paper we give a general definition in the context of abstract interpretation, show that arbitrary locality-based abstractions are hard to compute in general, and provide two solutions to this problem. The solutions are evaluated in several case studies.

1 Introduction

Consider a system acting on a set X of program variables over some value set V. An abstraction of the system, in the abstract-interpretation sense [1], deliberately loses information about the current values of the variables. Many abstractions can be intuitively visualized by imagining an observer who has access to the program code but is only allowed to retain limited knowledge about the values of the variables. For instance, the observer may only be allowed to retain the sign of a variable, its value modulo a number, or whether one value is larger than another one. In this paper we consider *locality-based* abstractions, which are best visualized by imagining a set of observers, each of which has a partial view of the system. Each observer has access to all the information 'within his window', but no information outside of it. For instance, in a system with three variables there could be three observers, each of them with perfect information about two of the variables, but no knowledge about the third. Given the set $\{\langle 1,1,0\rangle, \langle 1,0,1\rangle, \langle 0,1,1\rangle\}$ of valuations of the variables, the observer with access to, say, the first two variables 'sees' $\{\langle 1,1,\mathbf{u}\rangle, \langle 1,0,\mathbf{u}\rangle, \langle 0,1,\mathbf{u}\rangle\}$, where $\mathbf{u}$ stands for absence of information. Notice that information is lost: Even if the three observers exchange their informations, they cannot conclude that $\langle 1,1,1\rangle$ does *not* belong to the set of valuations.

The idea of local observers is particularly appropriate for distributed systems in which the value of a variable corresponds to the *local state* of a component of the system. In this case, a partial view corresponds to having no information from a number of components of the system. This is also the reason for the term "locality-based" abstraction.

⋆ The author wishes to thank the University of Stuttgart, where most of the work was done, for hospitality and both FRIA and FNRS for financial support.

Plan of the paper. In Sect. 3 we present a very general definition of locality-based abstraction, and study, in Sect. 4, the problem of computing the abstract *post*$^{\#}$ operator (i.e., the abstract operator corresponding to the usual *post* operator that computes the set of immediate successors on the concrete space). We observe that, in general, computing *post*$^{\#}$ involves solving an NP-complete problem, and present two orthogonal solutions to this problem in Sect. 5 and 6, respectively. Each of them leads to a polynomial-time algorithm. The first solution works for a restricted class of systems and arbitrary abstractions, while the second restricts the class of abstractions that are used but can be applied to arbitrary systems. In Sect. 7 we present an abstraction-refinement scheme which allows to progressively refine the precision of the abstractions while keeping good control of the time required to compute the $(post^{\#})^{*}$ operator, i.e., the operator yielding the set of reachable abstract states. Section 8 reports on experimental results obtained from an implementation of the approaches of Sect. 5 and 6.

Related work. Locality-based abstractions have been used before in the literature, but to the best of our knowledge not with the generality presented here. A particular case of locality-based abstraction are the *Cartesian abstractions* of [2], in which a set of tuples is approximated by the smallest Cartesian product containing this set. It corresponds to the case in which we have an observer for each variable (i.e., the observer can only see this variable, and nothing else). Another particular case that has been independently rediscovered several times is the *pairs* abstraction, in which we have an observer for each (unordered) pair of variables. In [3,4,5], this abstraction is used to overapproximate the pairs $\{l, l'\}$ of program points of a concurrent program such that during execution the control can simultaneously be at l, l'. In [6], it is used to overapproximate the pairs of places of a Petri net that can be simultaneously marked, and the abstraction is proved to be exact for the subclass of T-nets, also called marked graphs. In Graphplan, an approach to the solution of propositional planning problems [7,8], it is used to overapproximate the set of states reachable after at most n steps.

Prerequisites. The reader is expected to be familiar with the abstract interpretation framework and with the manipulation of symbolic data structures based on deterministic automata such as binary decision diagrams [9].

Full version. A version of the paper containing all proofs is available at `http://www.ulb.ac.be/di/ssd/cfv/publications.html`.

2 Preliminaries

System model. We fix a finite set V of *values* (in our examples we use $V = \{0, 1\}$). A *state* is a function $s\colon X \to V$, where $X = \{x_1, \ldots, x_n\}$ is a set of *state variables*. We also represent a state s by the tuple $(s[1], \ldots, s[n])$, where $s[i]$ is an abbreviation for $s(x_i)$. The set of all states over the set X of variables is denoted by $\mathscr{S}$.

Let X' be a disjoint copy of X. A *transition* t is a subset of $\mathscr{S} \times \mathscr{S}$, which we represent as a predicate $t(X, X')$, i.e., $(s, s') \in t$ if and only if $t(s, s')$ is true.

A *system* is a pair $Sys = (X, T)$ where X is a finite set of variables and T is a finite set of transitions. We define the *transition relation* $R \subseteq \mathscr{S} \times \mathscr{S}$ as the union of all the transitions of T.

Given a set of states S, we define the *successors* of S, denoted by $post[Sys](S)$, as the set of states s' such that $R(s, s')$ for some $s \in S$, and the *predecessors* of S, denoted by $pre[Sys](S)$, as the set of states s' such that $R(s', s)$ for some $s \in S$. We also write $post(S)$ or $pre(S)$ if the system Sys is clear from the context. We use the following notations: $post^0(S) = S$, $post^{i+1}(S) = post(post^i(S))$ for every $i \geq 0$, and $post^*(S) = \bigcup_{n \in \mathbb{N}} post^n(S)$. We use analogous notations for *pre*. A state s is *reachable* from S if $s \in post^*(S)$.

Partial states. Let $V^+ = V \cup \{\mathbf{u}\}$ where $\mathbf{u}$, disjoint from V, is the undefined value. It is convenient to define a partial order $\succeq$ on V^+, given by

$$v \succeq v' \overset{\text{def}}{\iff} (v' = \mathbf{u} \vee v = v') \enspace .$$

A *partial state* is a function $p \colon X \to V^+$. The set of all partial states is denoted by $\mathscr{P}$. The *support* of a partial state p is the set of indices $i \in \{1, \ldots, n\}$ such that $p[i] \neq \mathbf{u}$. We extend the partial order $\succeq$ to partial states:

$$p \succeq p' \overset{\text{def}}{\iff} \bigwedge_{x \in X} \big(p(x) \succeq p'(x)\big)$$

and to sets of partial states:

$$P \succeq P' \overset{\text{def}}{\iff} \forall p \in P \, \exists p' \in P' \colon p \succeq p' \enspace .$$

Given a partial state p, we define its *upward and downward closure* as $p\!\uparrow = \{p' \in \mathscr{P} \mid p' \succeq p\}$ and $p\!\downarrow = \{p' \in \mathscr{P} \mid p \succeq p'\}$, respectively. We extend these two notions to sets of partial states in the natural way. We say that P is *upward* or *downward* closed if $P\!\uparrow = P$ or $P\!\downarrow = P$, respectively. We also say that P is a *uc-set* or a *dc-set*.

Finally, we also define $p\!\Uparrow = p\!\uparrow \cap \mathscr{S}$, and extend the notation to sets of states.

3 Locality-Based Abstractions

Fix a system Sys and a set I of initial states of Sys. We say that a partial state p is *reachable* from I if *some* state $s \succeq p$ is reachable from I. Observe that with this definition p is reachable if and only if all partial states in the downward closure $p\!\downarrow$ are reachable. So the pieces of information we have about reachability of partial states can be identified with downward closed subsets of $\mathscr{P}$.

Assume now that the only dc-sets we have access to are those included in some dc-set $D \subseteq \mathscr{P}$, called in the rest of the paper a *domain.* If a state s is reachable, then all the elements of $s\!\downarrow \cap D$ are reachable by definition. However, the contrary does not necessarily hold, since we may have $s \notin D$. In our abstractions we overapproximate by declaring s reachable if all the elements of $s\!\downarrow \cap D$

are reachable, i.e., if all the information we have access to is compatible with s being reachable.

Intuitively, we can look at D as the union of sets $D_1, \ldots, D_n$, where all the partial states in D_i have the same support, i.e., a partial state $p \in D_i$ satisfies $p[i] = \mathbf{u}$ only if all partial states $p' \in D_i$ satisfy $p'[i] = \mathbf{u}$. The sets D_i correspond to the pieces of information that the different observers have access to. Notice that we can have a domain D_i like, say $D_i = \{\langle 0, 0, \mathbf{u}\rangle, \langle 1, 0, \mathbf{u}\rangle\}\!\downarrow$ in which the observer is only allowed to see some local states of the first two components, but not others, like $\langle 1, 1, \mathbf{u}\rangle$.

Recall that the powerset lattice $PL(A)$ associated to a set A is the complete lattice having the powerset of A as carrier, and union and intersection as least upper bound and greatest lower bound operations, respectively. In our abstractions the concrete lattice is the powerset lattice $PL(\mathscr{S})$ of the set of states $\mathscr{S}$.

We fix a domain $D \subseteq \mathscr{P}$, and define the *downward powerset lattice* $DPL(D)$ associated to D as the restriction of $PL(D)$ to the dc-sets included in D. That is, the carrier of $DPL(D)$ is the set of dc-subsets of D (which, since D is downward closed, contains D itself), and the least upper bound and greatest lower bound operations are union and intersection. Notice that $DPL(D)$ is well-defined because the union and intersection of a family of dc-sets is a dc-set. The abstract lattice of a locality-based abstraction is $DPL(D)$, and the concretization and abstraction mappings are defined as follows:

$$
\begin{aligned}
\alpha(S) &\overset{\text{def}}{=} S\!\downarrow \cap D && \text{for any } S \in PL(\mathscr{S}) \\
\gamma(P) &\overset{\text{def}}{=} \{s \in \mathscr{S} \mid s\!\downarrow \cap D \subseteq P\} && \text{for any } P \in DPL(D) \\
&= \mathscr{S} \setminus (D \setminus P)\!\Uparrow .
\end{aligned}
$$

Example 1. Consider the set of values V and the state variables X defined by $V = \{0, 1\}$ and $X = \{x_1, x_2, x_3\}$, respectively. The domain of pairs over X is given by

$$D_2 = \{(n, m, \mathbf{u}), (n, \mathbf{u}, m), (\mathbf{u}, n, m) \mid n, m \in \{0, 1\}\}\!\downarrow \ .$$

For the set $S = \{\langle 1, 1, 0\rangle, \langle 1, 0, 0\rangle, \langle 0, 1, 0\rangle\}$ we get

$$\alpha(S) = \{\langle 1, 1, \mathbf{u}\rangle, \langle 1, 0, \mathbf{u}\rangle, \langle 0, 1, \mathbf{u}\rangle, \langle 1, \mathbf{u}, 0\rangle, \langle 0, \mathbf{u}, 0\rangle, \langle \mathbf{u}, 1, 0\rangle, \langle \mathbf{u}, 0, 0\rangle\}\!\downarrow$$

and $(\gamma \circ \alpha)(S) = S$, i.e., in this case no information is lost.

Consider now the domain

$$D_1 = \{(n, \mathbf{u}, \mathbf{u}), (\mathbf{u}, n, \mathbf{u}), (\mathbf{u}, \mathbf{u}, n), (\mathbf{u}, \mathbf{u}, \mathbf{u}) \mid n \in \{0, 1\}\} \ .$$

In this case we get

$$
\begin{aligned}
(\gamma \circ \alpha)(S) &= \gamma(\{\langle 1, \mathbf{u}, \mathbf{u}\rangle, \langle 0, \mathbf{u}, \mathbf{u}\rangle, \langle \mathbf{u}, 1, \mathbf{u}\rangle, \langle \mathbf{u}, 0, \mathbf{u}\rangle, \langle \mathbf{u}, \mathbf{u}, 0\rangle, \langle \mathbf{u}, \mathbf{u}, \mathbf{u}\rangle\}) \\
&= \{0, 1\} \times \{0, 1\} \times \{0\}
\end{aligned}
$$

and in general $(\gamma \circ \alpha)(S)$ is the smallest cartesian product of subsets of V containing S, matching the cartesian abstractions of [2] [1].

Observe that for $D = \mathscr{P}$ we obtain

$$\begin{aligned} \alpha(S) &= S{\downarrow} && \text{for any } S \in PL(\mathscr{S}) \\ \gamma(P) &= P \cap \mathscr{S} && \text{for any } P \in DPL(D) \end{aligned}$$

and so $(\gamma \circ \alpha)(S) = S$, i.e., no information is lost.

The concrete $PL(\mathscr{S})$ and abstract $DPL(D)$ domains and the abstraction $\alpha\colon PL(\mathscr{S}) \mapsto DPL(D)$ and concretization $\gamma\colon DPL(D) \mapsto PL(\mathscr{S})$ maps form a *Galois connection*, denoted by $PL(\mathscr{S}) \underset{\gamma}{\overset{\alpha}{\rightleftharpoons}} DPL(D)$, for every domain D.

Proposition 1. *For every domain D, $PL(\mathscr{S}) \underset{\gamma}{\overset{\alpha}{\rightleftharpoons}} DPL(D)$.*

3.1 The *post*$^{\#}$ Operator

We define the function $post^{\#}[Sys, D]\colon DPL(D) \to DPL(D)$:

$$post^{\#}[Sys, D] \stackrel{\text{def}}{=} \lambda P.(\alpha \circ post[Sys] \circ \gamma)(P) \ .$$

We shorten $post^{\#}[Sys, D]$ to $post^{\#}$ if the system and the domain are clear from the context. We have the following characterization of $post^{\#}[Sys, D]$.

Proposition 2. *Let Sys and D be a system and a domain, respectively. For every $P \in DPL(D)$ and for every $p \in \mathscr{P}$*

$$p \in post^{\#}(P) \Longleftrightarrow p \in D \wedge \neg(pre(p{\Uparrow}) \subseteq (D \setminus P){\Uparrow}) \ .$$

Proof (of Proposition 2).

$$\begin{aligned} p \in post^{\#}(P) &\Leftrightarrow p \in (\alpha \circ post \circ \gamma)(P) \\ &\Leftrightarrow p \in D \wedge p \in ({\downarrow} \circ post \circ \gamma)(P) && \text{(Def. of } \alpha) \\ &\Leftrightarrow p \in D \wedge (p{\Uparrow} \cap (post \circ \gamma)(P) \neq \emptyset) \\ &\Leftrightarrow p \in D \wedge (pre(p{\Uparrow}) \cap \gamma(P) \neq \emptyset) \\ &\Leftrightarrow p \in D \wedge (pre(p{\Uparrow}) \cap \big(\mathscr{S} \setminus (D \setminus P){\Uparrow}\big) \neq \emptyset) && \text{(Def. of } \gamma) \\ &\Leftrightarrow p \in D \wedge \neg(pre(p{\Uparrow}) \subseteq (D \setminus P){\Uparrow}) && \square \end{aligned}$$

Using standard results of abstract interpretation we get for every set of states S that $(post^{\#})^*$ is a sound abstraction of $post^*$, i.e.:

$$post^*(S) \subseteq (\gamma \circ (post^{\#})^* \circ \alpha)(S) \qquad \text{for every } S \in PL(\mathscr{S}).$$

[1] Actually, the functions α and γ of [2] are slightly different, but their composition is the same as here.

4 The Complexity of Computing *post*$^{\#}$

In the rest of the paper we assume that sets of (partial) states are symbolically represented as *multi-valued decision diagrams* (MDD) (see [10] for more details) over the set of variables X with a fixed variable order. The cardinality of X will be denoted $|X|$. Given a set P of partial states we denote the MDD representing P by $P^{\mathcal{M}}$ and the size of $P^{\mathcal{M}}$ by $|P^{\mathcal{M}}|$. We also assume that each transition t of a system $Sys = (X, T)$ is symbolically represented as a MDD $t^{\mathcal{M}}$ over variables X, X' with a fixed variable order whose projection onto X coincides with the fixed order on X. The *size* of Sys is defined as $\sum_{t \in T} |t^{\mathcal{M}}| + |X|$ and denoted by $|Sys|$.

We consider the following decision problem.

Definition 1. *The problem POST$^{\#}$ is defined as follows:*
Instance: *a system $Sys = (X, T)$, an element $p \in D$ and two MDDs $D^{\mathcal{M}}$, $P^{\mathcal{M}}$, where D is a non-empty domain and $P \in DPL(D)$.*
Question: *$p \in post^{\#}[Sys, D](P)$?*

We say that a class of systems $\mathcal{C}$ is *polynomial* if the restriction $\mathrm{POST}^{\#}_{\mathcal{C}}$ of $\mathrm{POST}^{\#}$ to instances in which the system *Sys* belongs to $\mathcal{C}$ can be solved in polynomial time. Unfortunately, as we are going to show, unless P=NP holds, even very simple classes of systems are not polynomial. Before proceeding, we need a time complexity bound for some operation on MDD.

Proposition 3. *Let $p \in \mathscr{P}$ and $S^{\mathcal{M}}$ be a MDD for $S \subseteq \mathscr{P}$. We can decide in $O(|X| + |S^{\mathcal{M}}|)$ time if there exists $s \in S$ such that $p \succeq s$.*

Proof (of Proposition 3). We use a simple marking algorithm. Initially we mark the root node of $S^{\mathcal{M}}$. If a node m labelled by x_i is marked, we mark all successors n of m such that the edge $e = (n, m)$ is labelled with a $v_i \in V^+$ satisfying $p(x_i) \succeq v_i$. The state s exists iff at the end of the algorithm the accepting node is marked. □

The following proposition is proved by means of a simple reduction from the 3-colorability problem on graphs.

Proposition 4. *The following problem is NP-complete:*
Instance: *a set X of variables, and two MDDs $D^{\mathcal{M}}$, $P^{\mathcal{M}}$, where D is a non-empty domain on X and $P \in DPL(D)$.*
Question: *$\gamma(P) \neq \emptyset$?*
In particular, if P≠NP then there is no polynomial time algorithm to compute $\gamma(P)^{\mathcal{M}}$.

We are now able to present the main result of this section. Fix $V = \{0, 1\}$ and let $\{Sys_n\}_{n \geq 1}$ be the family of systems given by $Sys_n = (X_n, \{t_n\})$, $X_n = \{x_1, \ldots x_n\}$ and $t_n = \mathscr{S} \times \mathscr{S}$. Intuitively Sys_n is a system with n state variables and a unique transition t_n such that for any pairs of states s, s' we find that $(s, s') \in t_n$.

Proposition 5. *If the class $\mathcal{C} = \{Sys_n\}_{n \geq 1}$ of systems is polynomial, then P=NP.*

Proof. We reduce the problem of Prop. 4 to $\mathrm{POST}^{\#}_{\mathcal{C}}$. This shows that $\mathrm{POST}^{\#}_{\mathcal{C}}$ is NP-complete and so if $\mathcal{C}$ is polynomial, then P=NP.

Given an instance X, $D^{\mathcal{M}}$, $P^{\mathcal{M}}$ of the problem of Prop. 4, we build in polynomial time the partial state $\mathbf{u}^{|X|}$ ($\mathbf{u}^{|X|} \in D$ for any $D \neq \emptyset$) and the MDD $t^{\mathcal{M}}_{|X|}$ such that $t_{|X|} = \mathscr{S} \times \mathscr{S}$. The operator $post^{\#}$ is given by $(\alpha \circ post \circ \gamma)$ and so we have $\gamma(P) \neq \emptyset$ iff $\mathbf{u}^{|X|} \in post^{\#}[Sys_{|X|}, D](P)$. $\square$

This result shows that we do not have much hope of finding a broad, interesting class of polynomial systems. In the next sections we present two possible ways of dealing with this problem.

5 Partial Reachability

In this section we show that, if we change the concrete lattice in our abstractions by extending reachability also to partial states, then an interesting class of systems becomes polynomial. From now on, we assume the following ordering on $X = \{x_1, \ldots, x_n\}$ and its disjoint copy X': $x_1 < x'_1 < \cdots < x_n < x'_n$.

We define the notion of kernel of a transition. Intuitively, the kernel of a transition is the set of variables that are "involved" in it.

Definition 2. *Let $t(X, X')$ be a transition and let $Y \subseteq X$ be the smallest subset of X such that*

$$t(X, X') \equiv \hat{t}(Y, Y') \wedge \bigwedge_{x \in X \setminus Y} (x = x')$$

for some relation $\hat{t}$. We call $\hat{t}$ the kernel *of t, Y the* kernel variables *and $|Y|$ the* kernel width. *Given a partial state $p \in \mathscr{P}$, we denote by $\hat{p}$ the partial state given by*

$$\hat{p}[i] = \begin{cases} p[i] & \text{if } x_i \text{ belongs to the kernel variables of } t, \\ \mathbf{u} & \text{otherwise,} \end{cases}$$

and $\tilde{p}$ the partial state given by

$$\tilde{p}[i] = \begin{cases} p[i] & \text{if } x_i \text{ does not belong to the kernel variables of } t, \\ \mathbf{u} & \text{otherwise.} \end{cases}$$

We identify a partial state p and the pair $(\hat{p}, \tilde{p})$.

We need to extend the transitions to partial states.

Definition 3. *Let $Sys = (X, T)$ be a system and let $t \in T$. The* extended transition *$\underline{t} \subseteq \mathscr{P} \times \mathscr{P}$ is defined as follows:*

$$\underline{t}(p_1, p_2) \overset{\text{def}}{\iff} \exists p \in \mathscr{P} \colon \hat{t}(\hat{p}_1, p) \wedge p \succeq \hat{p}_2 \wedge \tilde{p}_1 = \tilde{p}_2 \ .$$

Given a system $Sys = (X, T)$, we define the extended transition relation *$\underline{R}$ of Sys as the union of all its extended transitions.*

The intuition behind this definition is as follows: If we know that p_1 is reachable (i.e., that some state $s \succeq p_1$ is reachable) and that $\underline{t}(p_1, p_2)$ holds, then we already have enough information to infer that p_2 is reachable. Let us see why. We know the values of all the variables involved in t (this is $\hat{p}_1$), and we know that we can reach $(p, \tilde{p}_1)$ from $(\hat{p}_1, \tilde{p}_1)$ (because $\hat{t}(\hat{p}_1, p)$). Now, since we can reach $(p, \tilde{p}_1)$ and we know that $p \succeq \hat{p}_2$ and $\tilde{p}_1 = \tilde{p}_2$, we can infer that p_2 is also reachable.

It is easy to show that the restriction of the extended reachability relation to states coincides with the reachability relation.

Lemma 1. *For any $t \in T$ and any $s_1, s_2 \in \mathscr{S}$, we have $t(s_1, s_2)$ iff $\underline{t}(s_1, s_2)$.*

Proof (of Lemma 1). Since s_2 is a state, $p \succeq \hat{s}_2$ holds if and only if $p = \hat{s}_2$, and so

$$\underline{t}(s_1, s_2) \Leftrightarrow (\hat{t}(\hat{s}_1, \hat{s}_2) \wedge \tilde{s}_1 = \tilde{s}_2) \Leftrightarrow t(s_1, s_2) \ . \qquad \square$$

In order to obtain a Galois connection, we extend the functions α, γ to $\underline{\alpha}$: $DPL(\mathscr{P}) \to DPL(D)$ and $\underline{\gamma}$: $DPL(D) \to DPL(\mathscr{P})$ in the obvious way:

$$\begin{aligned} \forall P \in DPL(\mathscr{P})\colon\ \underline{\alpha}(P) &\overset{\text{def}}{=} P{\downarrow} \cap D = P \cap D \quad (P \text{ is a } \textit{dc-set}) \\ \forall P \in DPL(D)\colon\ \underline{\gamma}(P) &\overset{\text{def}}{=} \{p \mid p{\downarrow} \cap D \subseteq P\} \\ &= \mathscr{P} \setminus (D \setminus P){\uparrow} \ . \end{aligned}$$

Proposition 6. *For every domain D, $DPL(\mathscr{P}) \underset{\underline{\gamma}}{\overset{\underline{\alpha}}{\rightleftarrows}} DPL(D)$.*

5.1 The $\underline{post}^{\#}$ Operator

We extend *post* and *pre* to $\underline{post}$ and $\underline{pre}$ on partial states by declaring $p' \in \underline{post}(p)$ and $p \in \underline{pre}(p')$ iff $\underline{R}(p, p')$. We have the following useful property:

Lemma 2. *Fix an arbitrary system, for every $p \in \mathscr{P}$, $\underline{pre}(p{\uparrow}) = \underline{pre}(p){\uparrow}$ and $\underline{post}(p{\uparrow}) = \underline{post}(p){\uparrow}$.*

The set $\underline{post}^{\#}[Sys, D](P)$ is given by

$$\{p_2 \in D \mid \exists p_1\colon \underline{R}(p_1, p_2) \wedge \neg\big(\exists p_3\colon p_3 \in (D \setminus P) \wedge (p_1 \succeq p_3)\big)\} \ .$$

Notice that, given MDDs $D^{\mathcal{M}}$, $P^{\mathcal{M}}$, $\underline{R}^{\mathcal{M}}$ and $\succeq^{\mathcal{M}}$, the set $\underline{post}^{\#}[Sys, D](P)$ can be computed symbolically using classical operations provided by any MDD package.

The following result, which makes use of Lemmata 1 and 2, shows that the $post^{\#}$ operator is a better approximation to *post* than $\underline{post}^{\#}$, i.e., replacing $post^{\#}$ by $\underline{post}^{\#}$ leads to a loss of precision.

Proposition 7. *Fix a system and a domain D. For every $P \in DPL(D)$, $post^{\#}(P) \subseteq \underline{post}^{\#}(P)$, but the converse does not hold.*

Proof. The first part is an easy consequence of the definitions, and can be found in the full version of the paper. Here we provide a detailed example proving the non inclusion of $\underline{post}^{\#}(P)$ in $\overline{post}^{\#}(P)$.

Fix $V = \{0, 1, 2\}$ and $Sys = (X, T)$ with $X = \{x_1, x_2, x_3, x_4\}$, $T = \{t_1, t_2, t_3, t_4\}$ and such that

$$\begin{array}{lll}
t_1(X, X') \equiv \hat{t}_1(Y, Y') \wedge x_3 = x_3' & \hat{t}_1 = \{(\langle 0,0,0\rangle, \langle 1,1,1\rangle)\} & Y = X \setminus \{x_3\} \\
t_2(X, X') \equiv \hat{t}_2(Y, Y') \wedge x_2 = x_2' & \hat{t}_2 = \{(\langle 0,0,0\rangle, \langle 1,1,2\rangle)\} & Y = X \setminus \{x_2\} \\
t_3(X, X') \equiv \hat{t}_3(Y, Y') \wedge \begin{pmatrix} x_1 = x_1' \\ x_4 = x_4' \end{pmatrix} & \hat{t}_3 = \{(\langle 0,0\rangle, \langle 1,1\rangle)\} & Y = \{x_2, x_3\} \\
t_4(X, X') \equiv \hat{t}_4(Y, Y') \wedge x_4 = x_4' & \hat{t}_4 = \{(\langle 1,1,1\rangle, \langle 2,2,2\rangle)\} & Y = X \setminus \{x_4\}
\end{array}$$

The domain D is the set of partial states $p \in \{0, 1, 2, \mathbf{u}\}^4$ such that for at most 2 indices i, j of $\{1, 2, 3, 4\}$: $p[i] \neq \mathbf{u}$ and $p[j] \neq \mathbf{u}$. The set of initial state I is given by $\{\langle 0, 0, 0, 0\rangle\}$. The set $(\underline{post}^{\#} \circ \underline{\alpha})(I\!\downarrow)$, denoted F, is given by

$$\begin{aligned}
F = \{&\langle 1,1,\mathbf{u},\mathbf{u}\rangle, \langle 1,\mathbf{u},\mathbf{u},1\rangle, \langle \mathbf{u},1,\mathbf{u},1\rangle, \langle 1,\mathbf{u},0,\mathbf{u}\rangle, \langle \mathbf{u},1,0,\mathbf{u}\rangle, \langle \mathbf{u},\mathbf{u},0,1\rangle \\
&\langle 1,\mathbf{u},1,\mathbf{u}\rangle, \langle 1,\mathbf{u},\mathbf{u},2\rangle, \langle \mathbf{u},\mathbf{u},1,2\rangle, \langle 1,0,\mathbf{u},\mathbf{u}\rangle, \langle \mathbf{u},0,1,\mathbf{u}\rangle, \langle \mathbf{u},0,\mathbf{u},2\rangle, \\
&\langle \mathbf{u},1,1,\mathbf{u}\rangle, \langle 0,1,\mathbf{u},\mathbf{u}\rangle, \langle \mathbf{u},1,\mathbf{u},0\rangle, \langle 0,\mathbf{u},1,\mathbf{u}\rangle, \langle \mathbf{u},\mathbf{u},1,0\rangle, \langle 0,\mathbf{u},\mathbf{u},0\rangle\}\!\downarrow .
\end{aligned}$$

It is routine to check that $(post^{\#} \circ \alpha)(I)$ and $(\underline{post}^{\#} \circ \underline{\alpha})(I\!\downarrow)$ coincide. Observe that $\langle 1, 1, 1, \mathbf{u}\rangle \in \underline{\gamma}(F)$ but

$$\{\langle 1,1,1,0\rangle, \langle 1,1,1,1\rangle, \langle 1,1,1,2\rangle\} \cap \underline{\gamma}(F) = \emptyset \ .$$

Now consider the second iteration. In this case we find that $\langle 2, 2, \mathbf{u}, \mathbf{u}\rangle \in \underline{post}^{\#}(F)$ but $\langle 2, 2, \mathbf{u}, \mathbf{u}\rangle \notin post^{\#}(F)$ which proves our claim. □

The loss of precision of $\underline{post}^{\#}$ is compensated by its better properties. We have the following characterization of $\underline{post}^{\#}[Sys, D](P)$.

Proposition 8. *Let Sys and D be a system and a domain, respectively. For every $P \in DPL(D)$, for every $p \in \mathscr{P}$*

$$p \in \underline{post}^{\#}(P) \Longleftrightarrow p \in D \wedge \neg(\underline{pre}(p) \succeq (D \setminus P)) \ .$$

Proof (of Proposition 8).

$$\begin{aligned}
p \in \underline{post}^{\#}(P) &\Leftrightarrow p \in (\underline{\alpha} \circ \underline{post} \circ \underline{\gamma})(P) \\
&\Leftrightarrow p \in D \wedge p \in (\underline{post} \circ \underline{\gamma})(P) && \text{(Def. of } \underline{\alpha}) \\
&\Leftrightarrow p \in D \wedge (\underline{pre}(p) \cap \underline{\gamma}(P) \neq \emptyset) \\
&\Leftrightarrow p \in D \wedge (\underline{pre}(p) \cap (\mathscr{P} \setminus (D \setminus P)\!\uparrow) \neq \emptyset) && \text{(Def. of } \underline{\gamma}) \\
&\Leftrightarrow p \in D \wedge \neg(\underline{pre}(p) \subseteq (D \setminus P)\!\uparrow) \\
&\Leftrightarrow p \in D \wedge \neg(\underline{pre}(p) \succeq (D \setminus P))
\end{aligned}$$

□

This proposition shows the difference between computing $post^{\#}$ and $\underline{post}^{\#}$: In the first case we have to deal with $(D \setminus P)\Uparrow$, which can have a much more complex symbolic representation than $(D \setminus P)$. In the case of $\underline{post}^{\#}$ we only need to deal with the set $(D \setminus P)$ itself.

5.2 The Complexity of Computing $\underline{post}^{\#}$

Given a system *Sys*, we define the problem $\underline{\mathrm{POST}}^{\#}$ as $\mathrm{POST}^{\#}$, just replacing $post^{\#}$ by $\underline{post}^{\#}$. As seen in Prop. 8, we can decide $\underline{\mathrm{POST}}^{\#}$ by checking whether $\underline{pre}(p) \succeq \overline{(D \setminus P)}$ holds. Consider the class of systems satisfying the following two conditions for every partial state p,

(a) $|\underline{pre}(p)|$ is bounded by a polynomial in $|X|$, and
(b) $\underline{pre}(p)^{\mathcal{M}}$ can be computed in polynomial time in $|X|$.

By Prop. 3, for $p' \in \underline{pre}(p)$, we can decide $\{p'\} \succeq (D \setminus P)$ in time $O(|D^{\mathcal{M}}| \cdot |P^{\mathcal{M}}| + |X|)$ and thus, given $\underline{pre}(p)^{\mathcal{M}}$, $D^{\mathcal{M}}$, and $P^{\mathcal{M}}$, we can decide $\underline{pre}(p) \succeq (D \setminus P)$ in polynomial time. Since $|\underline{pre}(p)^{\mathcal{M}}|$ is polynomial in $|X|$ and $|X|$ is $O(|Sys|)$, we can decide $\underline{\mathrm{POST}}^{\#}$ in polynomial time. It follows that these systems are polynomial for $\underline{\mathrm{POST}}^{\#}$.

We now show that an interesting class of systems satisfies (a) and (b). Intuitively, we look at a system on a set X as a set having $|X|$ components. Each variable describes the local state of the corresponding component.

Definition 4. *A system $Sys = (X, T)$ is k-*bounded *if the width of the kernel of all transitions of T is bounded by k.*

Loosely speaking, a system is k-bounded if its transitions involve at most k components. Many systems are k-bounded. For instance, consider systems communicating by point to point channels. If we describe the local state of a component/channel by one variable, then usually we have $k = 2$, because a transition depends on the current state of the receiving/sending component and on the state of the channel. Another example are token ring protocols, where each component communicates only with its left and right neighbours. These systems are at most 3-bounded.

Observe that each k-bounded system is equivalent to another one satisfying $|T| \leq |X|^k$: if there is $\hat{t}_i(Y_i, Y_i'), \hat{t}_j(Y_j, Y_j')$ such that $i \neq j$ but $Y_i = Y_j$, then we can replace $\hat{t}_i$ and $\hat{t}_j$ by $(\hat{t}_i \vee \hat{t}_j)(Y_i, Y_i')$.

Proposition 9. *Let p be a partial state of a k-bounded system. The set $\underline{pre}(p)$ contains at most $|X|^k \cdot |V^+|^k$ elements, and $\underline{pre}(p)^{\mathcal{M}}$ can be computed in time polynomial in $|X|$.*

Corollary 1. *For a fixed $k \geq 0$, the class of k-bounded systems is polynomial.*

6 Neighbourhood Domains

The polynomiality result of the last section is obtained at a price: we had to consider less precise abstractions, and we had to restrict ourselves to k-bounded systems. In this section we define an approach, applicable to arbitrary systems, that uses a class of domains called *neighbourhood domains.* Intuitively, in a neighbourhood domain the variables an observer has access to must be *neighbours* with respect to the order used to construct the MDDs. E.g., an observer may observe variables x_3, x_4, x_5, but not x_1, x_8.

We say that a class $\mathcal{D}$ of domains is *polynomial* if the restriction $\mathrm{POST}^{\#}_{\mathcal{D}}$ of $\mathrm{POST}^{\#}$ to instances in which the domain D belongs to $\mathcal{D}$ can be solved in polynomial time.

By Prop. 4 we know that, unless P=NP, there is no polynomial algorithm to compute $\gamma(\cdot)$. We define hereafter a class of domains which avoids this problem, i.e., for every set P in the domain, the $\gamma(P)^{\mathcal{M}}$ can be computed in time polynomial in $|X|$, $|P^{\mathcal{M}}|$ and $|D^{\mathcal{M}}|$.

Definition 5. *Let $x_1 < \cdots < x_n$ be a variable ordering for X and let $1 \leq k \leq |X|$. The k-*neighbourhood *domain D is defined as follows*

$$D(X) \equiv \bigvee_{V_i \in \mathcal{V}} \Big(\bigwedge_{x \in X \setminus V_i} (x = \mathbf{u}) \Big)$$

where $\mathcal{V}$ is the set of all the sets of k consecutive variables, e.g., for $n \geq k+2$ we find that $\{x_2, \ldots, x_{2+k}\} \in \mathcal{V}$.

In what follows, we sometimes abbreviate $\bigwedge_{x \in X \setminus V_i}(x = \mathbf{u})$ to $D_i(X)$.

The following two propositions introduce the two key properties of neighbourhood domains:

Proposition 10. *Let D be a k-neighbourhood domain D, and let $P \in DPL(D)$. The MDD for the set $(D \setminus P)\uparrow^{\mathcal{M}}$ can be be computed in polynomial time in $|(D \setminus P)^{\mathcal{M}}|$ (and so, in particular, it is only polynomially larger than $(D \setminus P)^{\mathcal{M}}$).*

We prove a similar result for the computation of the downward closure.

Proposition 11. *Given a k-neighbourhood domain D and a set $S \subseteq \mathscr{S}$, the MDD for $(S\downarrow \cap D_i)$ with $V_i \in \mathcal{V}$ can be computed in polynomial time in $|D_i^{\mathcal{M}}| \cdot |S^{\mathcal{M}}|$.*

It follows that, for neighbourhood domains, both α and γ can be computed in polynomial time in their input size.

Proposition 12. *For a fixed $k \geq 0$, the class of k-neighbourhood domains is polynomial.*

Proof. Consider an instance of $\mathrm{POST}^{\#}$ in which D is a k-neighbourhood domain. We give a polynomial algorithm to decide if $p \in post^{\#}[Sys, D](P)$. By the definition of $post^{\#}$ and α, we have $p \in post^{\#}[Sys, D](P)$ iff there is a transition

t and states s, s' such that $s \in \gamma(P) \wedge t(s, s') \wedge s' \in p\Uparrow$. By Prop. 10, $\gamma(P)^{\mathcal{M}}$ over variables X can be computed in polynomial time in $|D^{\mathcal{M}}|$ and $|P^{\mathcal{M}}|$, and an MDD $p\Uparrow^{\mathcal{M}}$, over variables X' can be computed in polynomial time in $|X|$. The algorithm constructs, for each transition t of the system, an MDD for the formula $\gamma(P)^{\mathcal{M}} \wedge t(X, X') \wedge p\Uparrow^{\mathcal{M}}(X')$ and checks if it encodes the empty set. Since the construction and the check can be carried out in polynomial time, we are done. $\square$

Moreover, while in the concrete system the number of image computations may also be exponential, here we get a much better bound. Given a k-neighbourhood domain, each of the $(|X| - (k-1))$ formulæ $D_i(X)$ has exactly $|V|^k$ satisfying partial states. This leads us to the following fact: for any k-neighbourhood domain, for any system and for any set I of initial states, the number of iterations required to reach the fixed point in the computation of $(post^{\#})^*(I)$ is bounded by $(|X| - (k-1)) \times |V|^k$. Choosing the domain adequately, we thus have a way to control the complexity of computing $(post^{\#})^*(I)$. In practice this suggests the following strategy: if the *post* image computation is costly we can reduce the number of iterations needed to reach the fixed point by choosing a k-neighbourhood domain with $k << |X|$, of course at the prize of losing precision.

7 Abstraction Refinement

In this section, we describe an abstraction-refinement loop for testing reachability using the partial-reachability method. Given a system $Sys = (X, T)$, a set of initial states I, and a partial state u. Our goal is to check whether u is reachable, i.e. whether $u\Uparrow \cap post^*(I) \neq \emptyset$.

Our method starts from a (given) initial domain D and computes the reachable states in the abstraction, i.e. $(\underline{post}^{\#})^*(I\downarrow)$. If the latter includes u, we check if the imprecision caused by choosing the domain D might be responsible for the positive result. If so, we refine D accordingly.

More precisely, our scheme consists of the following two steps:

Search. Compute the sequence $F_0 = \underline{\alpha}(I\downarrow)$, $In_0 = Out_0 = \emptyset$, and then for $i \geq 0$:

$$\begin{aligned} T_{i+1} &= \{\, (p, p') \mid p \in \underline{\gamma}(F_i),\ p' \in (\underline{\alpha} \circ \underline{post})(p) \,\} \\ F_{i+1} &= F_i \cup \{\, p' \mid \exists p \colon (p, p') \in T_{i+1} \,\} \\ In_{i+1} &= In_i \cup \{\, (p, p') \in T_{i+1} \mid p \in D \,\} \\ Out_{i+1} &= Out_i \cup \{\, (p, p') \in T_{i+1} \mid p \in \mathscr{P} \setminus D \,\}. \end{aligned}$$

Stop when the sequence of F_is reaches a fixed point. We denote the values of the sets in the fixed point as F_f, In_f, Out_f, respectively.

Notice that T_i records a reachability relation between partial states, where the left components can be either previously computed partial states in D or partial states whose reachability was (potentially wrongly) 'inferred' by the concretization. We then have $F_{i+1} = F_i \cup \underline{post}^{\#}(F_i)$. The sequence of In sets records the reachability relation between states for which no inference was used, whereas *Out* records the relations for which inference was used, i.e. the places where potential imprecision was introduced.

By Prop. 8, the T_i sets can be computed efficiently.

Refine. If $u \notin \underline{\gamma}(F_f)$, then by Prop. 7 u is unreachable, and we stop with a negative result.

Otherwise, if $u \notin D$, then we inferred reachability of u from the reachability of several partial states. We then refine D to $D \cup u{\downarrow}$, which forces the next iteration to 'watch' the partial state u explicitly. (Notice that we could have done this before the first iteration, but then again we might be able to refute reachability of u in the first iteration without doing this.)

Failing both tests, we check whether there is a real trace from an initial state to u. For this, we compute backwards reachability using the relation In_f. We conclude that u is reachable in the concrete system if $\exists i \in \underline{\alpha}(I{\downarrow})\colon (i,u) \in In_f^*$. Executing this check step for step also gives us the ability to output a witness path for u's reachability.

Otherwise, u was reachable in the abstraction because of a step contained in Out_f. To prove concrete reachability of u, we must prove that the partial source states of these steps were indeed reachable. Thus, we compute the set $A = \{\, p \mid In_f^*(p,u) \,\}$ and then refine D to $D \cup \{\, p \mid \exists p' \in A\colon (p,p') \in Out_f \}$.

Our approach is different from the usual CEGAR approach (see [11] for more details), where one tests whether an abstract counterexample found in the search phase is spurious. If it is, one refines the abstraction to prevent that counterexample from being found again. In our approach, we cannot tell whether a counterexample is spurious or not; we can merely test whether potentially imprecise information was used. If the counterexample was spurious, our refinement prevents it from being found again. If the counterexample was real, then our refinement gathers additional information to prove the counterexample correct.

Extensive work (see [12,13,14]) investigates the connection between abstract model checking, and in particular the CEGAR approach, and the domain refinement used in abstract interpretation. As a future work we plan to investigate the connection but relatively to the refinement technique we proposed in this section.

8 Experiments

We have produced a prototype implementation of the approaches of Sect. 5 (with abstraction refinement) and 6 (without abstraction refinement), and applied it to two well-known examples. The examples only use boolean variables, and so we use BDDs instead of MDDs. Since our implementation is preliminary and our main motivation is to provide a space-efficient method, we only report on the sizes of the BDDs used to decide a property. We compare them with the BDD size of the full set of reachable states, which is computed using NuSMV [15].

8.1 Dining Philosophers Example

Our first example is a deterministic non-symmetric solution to the dining philosophers problem taken from [16]. The model uses two arrays, one for the forks and the other for the philosophers, both of size N, the number of philosophers. Each fork is represented by two bits, and each philosopher by three. For our experiments, we use two different 'natural' variable orders.

(A) The first order puts the bits for the forks at the top and the philosophers at the bottom, while each array element is stored with its most significant bit at the top.
(B) The second order interleaves forks and philosophers, i.e. we put the first fork at the top, then the first philosopher, then the second fork etc.

The sizes of the BDDs encoding the full set of reachable states are listed (for orders A and B and various values of N) in the left half of Table 1. As can be seen, they strongly depend on the variable ordering, with order B working far better.

We consider the following three properties:

1. Is it possible that two neighbouring philosophers eat at the same time? (This property is false in the model.)
2. Is it possible for all forks to be taken at the same time? (This property is true in the model.)
3. Is it possible for philosophers 1 and 3 to eat at the same time? (This property is true for all $N > 3$.)

Notice that, without a refinement loop, an abstraction can only prove that a set of states is *not* reachable, and so it can only be used to decide property 1. Since we have not implemented the refinement loop for the neighbourhood approach (see Sect. 6), we only apply it to this property. The partial-reachability approach is applied to all three properties.

In the neighbourhood approach, we can decide property 1 by taking $N+1$ and $k = 3$ for ordering A and B, respectively. The BDD sizes for $(post^{\#})^*(I)$ are shown in the right half of Table 1. We observe that, as for full reachability, the BDDs grow exponentially for ordering A and only linearly for ordering B.

Table 1. BDD sizes in Dining Philosophers example, part 1

	full reachability		neighb. approach	
N	ord. A	ord. B	ord. A	ord. B
2	26	25	13	18
3	64	41	40	36
4	140	57	82	54
7	1,204	105	304	108
10	9,716	153	670	162
15	311,284	273	1,600	252

Table 2. Results for Dining Philosophers, partial-reachability approach

	starting with 1 component						starting with 2 components					
	prop. 1		prop. 2		prop. 3		prop. 1		prop. 2		prop. 3	
N	$\|post^*\|$	$\#ref$	$\|post^*\|$	$\#ref$	$\|post^*\|$	$\#ref$	$\|post^*\|$	$\#ref$	$\|post^*\|$	$\#ref$	$\|post^*\|$	$\#ref$
2	52	5	34	3	n/a	n/a	38	0	46	2	n/a	n/a
3	128	4	112	4	109	7	73	0	144	4	73	0
4	217	4	321	5	89	5	112	0	426	5	152	4
7	523	4	6,781	8	167	5	259	0	6,300	8	335	4
10	919	4	??	?	245	5	451	0	??	?	563	4
15	1,807	4	??	?	375	5	871	0	??	?	1,043	4

However, the constant of the growth for ordering A is much smaller, i.e., the approach is far less sensitive to the variable order.

The results for the partial reachability approach are detailed in Table 2. We considered two different initial abstractions for the refinement loop. In the first one, we take one observer for each component (philosopher or fork); in the second, one observer for each pair of components (left and right part of Table 2, resp.). The $\#ref$ columns denote the number of refinements that were necessary to prove or disprove the properties. The column marked $|post^*|$ gives the number of BDD nodes used to represent $(\underline{post}^{\#})^*(I{\downarrow})$ in the last refinement, where this number was highest. The representation of D was either nearly of the same size or significantly lower. The data for the orderings A and B are almost identical, and so only those for ordering A are shown. For properties 1 and 3, we observe the same pattern as in the neighbourhood case: the approach works well and is far less sensitive to the variable ordering. Looking closely, we observe that the 2-component initialization works better for property 1, presumably because the property is a conjunction of sub-properties concerning pairs of philosophers. For property 3, the 1-component initialization works better, probably because it concerns only 2 specific components. Property 2 is a case in which the locality-based approach works far worse than full reachability: The property is universally quantified, forcing the abstraction refinement to consider tuples ranging over all components.

8.2 Production Cell Example

Our second example is a model of the well-known production cell case study taken from [17]. Our encoding of the model has 15 variables with 39 bits altogether. We tested all fifteen safety properties mentioned in [17], but present the results for a few representative ones (the rest yielded similar results). The results are shown in Table 3.

Table 3 lists results for instantiations of the model with one and five plates. The number $|reach|$ is the BDD size of the reachable state space as computed by NuSMV, while $|post^*|$ and $\#ref$ have the same meanings as in Table 2.

The results show that while the reachable state space grows (linearly) with the number of plates, the partial-reachability approach is largely unaffected by their number. Moreover, while the number of refinement iterations varies (the

Table 3. Results for production cell example

	One plate		Five plates	
	$\lvert reach\rvert = 230$		$\lvert reach\rvert = 632$	
Prop	$\lvert post^*\rvert$	$\#ref$	$\lvert post^*\rvert$	$\#ref$
1	83	2	83	2
2	88	4	92	6
4	76	1	76	1
6	105	5	120	8
11	146	3	146	3

largest number of refinements was 13), the BDD sizes vary only by about 50% between the smallest and the largest example. As the number of plates increases, the space savings of the locality-based approach become significant.

In the neighbourhood approach, 4 out of the 15 properties could be proved with a neighbourhood domain of size $k = 2$. Independently of the number of plates, the number of BDD nodes representing the reachable state space was 129. A domain of size $k = 3$ was sufficient to verify another 7 properties; the number of BDD nodes increased to 208. The remaining properties could only be verified using full reachability, i.e. the neighbourhood approach did not have any advantage in this case.

9 Conclusions

We have presented locality-based abstractions, in which a state of the system is abstracted to the collection of views that some observers have of the state. Each observer has only access to some variables of the system. As pointed out in the introduction, special cases of locality-abstractions have been used in different contexts (planning, analysis of concurrent programs, concurrency theory). In this paper we have (1) generalized the abstractions used in other papers, (2) put them in the framework of abstract interpretation, (3) pointed out the bad complexity of the computation of the abstract successor operator for arbitrary locality-based abstractions, (4) provided two efficient solutions to this problem, and (5) evaluated these solutions on a number of examples. Our conclusion is that locality-based abstractions are a useful tool for the analysis of concurrent systems.

In our approach we have assumed that variables have a finite domain, and that if an observer has access to a variable, then it gets full information about its value. Both assumptions can be relaxed. For instance, locality-based abstractions can be easily combined with any of the usual abstractions on integer variables. It must only be required that clustered variables must be observable by the same observer.

References

1. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. POPL, ACM Press (1977) 238–252
2. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and cartesian abstraction for model checking C programs. In: Proc. TACAS. (2001) 268–283
3. Naumovich, G., Avrunin, G.S.: A conservative data flow algorithm for detecting all pairs of statements that may happen in parallel. In: Proc. FSE. Volume 23, 6 of Software Engineering Notes., ACM Press (1998) 24–34
4. Naumovich, G., Avrunin, G.S., Clarke, L.A.: An efficient algorithm for computing *mhp* information for concurrent Java programs. In: Proc. FSE. Volume 1687 of LNCS. (1999) 338–354
5. Naumovich, G., Avrunin, G.S., Clarke, L.A.: Data flow analysis for checking properties of concurrent Java programs. In: Proc. ICSE, ACM Press (1999) 399–410
6. Kovalyov, A.: Concurrency relations and the safety problem for petri nets. In: Proc. ATPN. Volume 616 of LNCS. (1992) 299–309
7. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 279–298
8. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. In: Proc. IJCAI. (1995) 1636–1642
9. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35** (1986) 677–691
10. Srinivasan, A., Kam, T., Malik, S., Brayton, R.K.: Algorithms for discrete function manipulation. In: IEEE/ACM ICCAD. (1990) 92–95
11. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50** (2003) 752–794
12. Ranzato, F., Tapparo, F.: Making abstract model checking strongly preserving. In: Proc. SAS. Volume 2477 of LNCS. (2002) 411–427
13. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples, and refinements in abstract model-checking. In: Proc. SAS. Volume 2126 of LNCS. (2001) 356–373
14. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. J. ACM **47** (2000) 361–416
15. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV version 2: An opensource tool for symbolic model checking. In: Proc. CAV. Volume 2404 of LNCS. (2002)
16. Zuck, L.D., Pnueli, A., Kesten, Y.: Automatic verification of probabilistic free choice. In: Proc. VMCAI. Volume 2294 of LNCS. (2002) 208–224
17. Heiner, M., Deussen, P.: Petri net based qualitative analysis - a case study. Technical Report I-08/1995, Brandenburg Tech. Univ., Cottbus (1995)