On the Strength of Owicki-Gries for Resources

Alexander Malkis and Laurent Mauborgne

IMDEA Software Institute

Abstract. In multithreaded programs data are often separated into lock-protected resources. Properties of those resources are typically verified by modular, Owicki-Gries-like methods. The modularity of the Owicki-Gries method has its price: proving some properties may require manual introduction of auxiliary variables. What properties can be proven without the burden of introducing auxiliary variables? We answer this question in the abstract interpretation framework. On one hand, we reveal a lattice structure of the method and supply a syntax-based abstract transformer that describes the method *exactly*. On the other hand, we bound the loss of precision from *above* and *below* by transition-relation-independent weakly relational closures. On infinitely many programs the closures coincide and describe the precision loss exactly; in general, the bounds are strict. We prove the absence of a general exact closure-based fixpoint characterization of the accuracy of the Owicki-Gries method, both in the collecting semantics and in certain trace semantics.

1 Introduction

The paper will characterize the accuracy of a popular verification method for proving safety properties of concurrent programs that operate on resources.

A program operating on resources is a multithreaded program in which threads communicate via sequentially consistent shared memory and in which all shared variables are partitioned into disjoint resources. Each resource may be *available* or *busy*. To access a variable belonging to a resource, a thread waits until the resource gets available and then starts a critical section for that resource, thus making the resource busy. While staying in the critical section, the thread can read and write the resource data, and no other thread can enter a critical section for the same resource, so no other thread can access the resource data. After the thread finishes accessing the resource, it exits the corresponding critical section, making the resource available.

Simple safety properties of such programs can be proven modularly. A modular proof of a program consists of an annotation per control flow location and an annotation per resource. Roughly, an annotation of a control flow location describes the valuations of the variables that the thread may access at that location. An annotation of a resource describes, roughly, the state of the resource when it is available. The annotations of locations have to be sequentially consistent similarly to the standard Hoare-style assertional logic, but with two changes: when a thread acquires a resource, the proof may assume that the invariant of the acquired resource holds, and on releasing a resource the resource invariant should be reestablished.

Such modularity incurs a loss of precision: for many programs, too strong but valid properties cannot be proven by the method. To prove such properties, the program can be manually augmented with so-called auxiliary variables. A set of variables is called auxiliary if, intuitively, projecting the traces of the program to the other variables gives the same result as removing all the statements involving the auxiliary variables and projecting afterwards. A modular proof is created for the augmented program, then the proven property is projected onto the original variables. The ability to use auxiliary variables makes the proof system complete. So far, auxiliary variables have been introduced purely manually, while the construction of the remaining proof can be automated.

We will estimate the loss of precision inherent to the *core* of the Owicki-Gries method, which consists of all the proof rules of the Owicki-Gries logic except the ability to use auxiliary variables, in abstract interpretation.

We will show a rich lattice structure of the Owicki-Gries-core proofs and a syntax-based abstract transformer describing the Owicki-Gries core *exactly*.

We will observe that there is no transition-relation-independent approximation operator that exactly characterizes the loss of precision of the Owicki-Gries core in the collecting semantics. We will also note the absence of equivalent trace-based characterizations of certain kinds.

We will find an approximation operator that induces a useful upper bound on this set, i.e., that will describe how much precision the Owicki-Gries core always loses. This approximation operator depends only on the syntactic structure of the program, but not on its exact transition relation. If a property is provable by the Owicki-Gries core, it is provable by abstract interpretation with this approximation.

We will present an approximation operator inducing a lower bound on the set, i.e., describing how much precision the Owicki-Gries core always retains. This approximation operator also depends only on the syntactic structure of the program, but not on its exact transition relation. If a property is provable by abstract interpretation with this approximation, it is provable by the Owicki-Gries core.

In passing, we will demonstrate an infinite class of simple programs for which both bounds are equal and a program on which both bounds are strict.

In short, the main results and their position are depicted in Fig. 1.

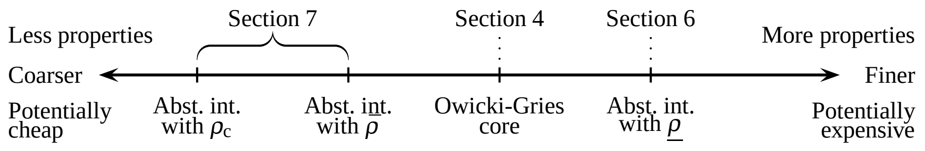

Fig. 1: Precision of analyses. The upper bound is $\underline{\rho}$, the lower bound is $\overline{\rho}$. In addition to $\overline{\rho}$, we will consider a simpler lower bound ρ_c, which is coarser than $\overline{\rho}$, but easier to understand.

The rigorous formalization, computations, proofs, recommendations for practitioners, and comparison to abstract interpretations for general programs are found in the appendix of the technical report [12].

2 Resource-manipulating language and its semantics

2.1 Language RPL

Now we briefly recall the parallel language RPL (Restricted Parallel Language, rigorously defined in [15]) in which shared data are separated into resources and access to a resource is granted exclusively.

Let threads be indexed by the elements of the set *Tid* and, without loss of generality, start their executions together. Threads operate on data variables from the set *DataVar*. A *resource* is a set of data variables; the resources are disjoint. Let *Res* be the set of all resources.

A *statement* is either

- an atomic statement, i.e., an assignment or an "`assume` ϕ" (which skips if the argument expression evaluates to *true* and blocks otherwise),
- a sequential composition of two statements, an `if-then-else`, or a `while` loop,
- a critical section `with` r `when` ϕ `do` C `endwith` where r is a resource, ϕ a formula over data variables and C some statement. We write `with` r `do` C `endwith` when ϕ is `true`.

A thread executes a statement.

A data variable is called *local* to thread t if all the assignments to the variable are syntactically in t. If a variable appears in thread t, it should belong to a resource or be local to t. If a variable belongs to a resource r, it can appear only inside a critical section for r, i.e., inside a "`with` r..." statement.

Each resource r is associated with a set of *proof variables* $\mathit{ProofVar}(r)$, which are all data variables that are not assigned except maybe in critical sections for r. All our examples will satisfy $\mathit{ProofVar}(r) = r$ (in general, $\mathit{ProofVar}(r) \supseteq r$).

An RPL program is described by a set of threads, a set of resources and an initial condition on the variables.

2.2 Semantics of RPL

To connect to abstract interpretation, we describe programs in RPL as transition systems.

For each thread t we introduce a fresh non-data program counter variable pc_t, which

- is local to thread t, but not to any other thread, and
- doesn't belong to any resource, and
- takes values from the set of control flow locations *PL* (Program Location).

Let *PCVar* be the set of all program counter variables and $\mathit{Var} = \mathit{DataVar} \dot{\cup} \mathit{PCVar}$.

We associate each control flow point $p \in \mathit{PL}$ with a set $\mathit{rsc}(p)$ of resources r such that p is inside a `with` r... statement.

We allow the same control flow location, say, p, to occur in different threads.

A state of a program is a map from *Var* to the set of values *Val* that includes *PL* and the ranges of all data variables. We are going to speak about *consistent states* only: those are states that

- map program counters to elements of *PL* and
- satisfy mutual exclusion: the sets of resources held by different threads are disjoint.

A thread t *holds* a resource r in a consistent state v if $r \in rsc(v(pc_t))$. A resource r is *busy* in a consistent state v if some thread holds it in v, otherwise r is *available* in v.

Each statement of thread t induces a set of transitions $v \rightarrow_t v'$ where v, v' are consistent states. The transitions induced by the non-`with` statements are straightforward. The statement `with` r `when` ϕ `do` C blocks when r is busy or when ϕ does not hold, otherwise it changes the control flow location to the initial control flow location of C; going out of the critical section is an unconditional change of the control flow location.

Let *init* be the set of initial states corresponding to the initial condition and let the successor map

$$post(S) \; = \; \{v' \mid \exists\, v \in S,\, t \in Tid \colon v \rightarrow_t v'\}$$

return the set of all one-step successors of a set of consistent states.

The *syntactic structure* of an RPL program is given by: thread identifiers, control flow locations, values, resources, locals and program counter variables of the threads, proof variables, map *rsc*. A syntactic structure is just a tuple of basic mathematical objects (sets, variables, maps). It also exists independently of an RPL program.

3 Owicki-Gries-core proofs

3.1 Lattice of Owicki-Gries-core proofs

Now we formalize the core of the Owicki-Gries proof system. We abstract away from the details of logic, handling sets of variable valuations instead of formulas.

Fix an arbitrary syntactic structure.

A *program annotation* (I, M) consists of

- a resource invariant I_r for each resource r, which is a set of valuations of proof variables of r,
- a control flow annotation $M_{p,t}$ for each control flow location p and each thread t, containing valuations of data variables which are local to t or which are proof variables of a resource held at p.

We say that a consistent state v satisfies I_r (resp. $M_{p,t}$), if the projection of v to $ProofVar(r)$ (resp. to local data variables of t and all proof variables of all resources in $rsc(p)$) is in I_r (resp. in $M_{p,t}$).

A program annotation (I, M) *denotes* the set of all consistent states v such that

- for each resource r which is available in v, the state v satisfies I_r, and
- for each thread t, the state v satisfies $M_{v(pc_t),t}$.

We define $\gamma_{\mathrm{OG}}(I, M)$ as the set of all consistent states denoted by (I, M).

Now fix an arbitrary RPL program obeying the given syntactic structure.

An *Owicki-Gries-core proof* of the program is a program annotation that satisfies the following conditions.

- It is *sequentially consistent*, which means the following: consider any transition $v \rightarrow_t v'$ from a control flow location p to a control flow location p' such that v satisfies $M_{p,t}$. There are three cases.
    - The transition does not cross the border of a critical section. Then v' should satisfy $M_{p',t}$.

  - The transition starts a critical section of a resource r. Additionally, assume that v satisfies I_r. Then v' should satisfy $M_{p',t}$.
  - The transition finishes a critical section of a resource r. Then v' should satisfy both $M_{p',t}$ and I_r.
– It admits the initial states, i.e., denotes a superset of *init*.

These Hoare-style rules treat critical sections specially. First, on entering a critical section, we additionally assume the resource invariant. Second, on leaving a critical section, we should prove the resource invariant.

Every Owicki-Gries-core proof denotes an inductive invariant. A set of consistent states S is *Owicki-Gries-core provable* if there is an Owicki-Gries-core proof that denotes a subset of S.

Interestingly, the set of program annotations of a fixed program forms a complete lattice with respect to componentwise inclusion order. Restricting this order to the set of Owicki-Gries-core proofs gives a complete lattice of Owicki-Gries-core proofs, even a Moore family. So each program has a unique smallest Owicki-Gries-core proof. This proof denotes the strongest Owicki-Gries-core-provable property. To investigate the power of the method, we will look at the smallest Owicki-Gries-core proofs and strongest Owicki-Gries-core-provable properties.

3.2 Examples

Now we will look at examples of programs with their smallest Owicki-Gries-core proofs. On Readers-Writers, as well as for a whole class of similarly simple programs, all proof methods will have the same precision, on Upper all proof methods will have different precision and on the class SepThreads no precision loss will be observed at all.

We'll use Owicki-style notation. In particular, "`resource` $control(ww, ar, aw)$" means that the resource named *control* contains exactly *ww*, *ar* and *aw*.

Example 1 (Readers-Writers). A number of threads share a file simultaneously: $n > 1$ need reading access and $m > 1$ writing access. Any number of readers may access the file simultaneously, but a writer must have exclusive access. In Fig. 2, all the data variables range over nonnegative integers by default, subtracting a positive value from 0 blocks.

```
initially $ww = ar = aw = 0$
resource  $control(ww, ar, aw)$

reader$_1$  ||  ...  ||  reader$_n$  ||  writer$_1$  ||  ...  ||  writer$_m$
```

```
// reader$_i$:
                 while $true$ do
startreadA:         $\{true\}$
                    with $control$
                    when $ww = 0$ do
startreadB:            $\{ww = 0 \wedge aw \leq 1\}$
                       $ar := ar + 1$
startreadC:            $\{ww = 0 \wedge aw \leq 1 \leq ar\}$
                    endwith;
read:               $\{true\}$
                    ;
finishreadA:        $\{true\}$
                    with $control$ do
finishreadB:           $\{aw \leq 1\}$
                       $ar := ar - 1$
finishreadC:           $\{aw \leq 1\}$
                    endwith
                 endwhile
```

```
// writer$_j$:
                 while $true$ do
askwriteA:          $\{true\}$
                    with $control$ do
askwriteB:             $\{aw \leq 1\}$
                       $ww := ww + 1$
askwriteC:             $\{aw \leq 1 \leq ww\}$
                    endwith;
startwriteA:        $\{true\}$
                    with $control$
                    when $ar = aw = 0$ do
startwriteB:           $\{ar = aw = 0\}$
                       $aw := aw + 1$
startwriteC:           $\{ar = 0 \wedge aw = 1\}$
                    endwith;
write:              $\{true\}$
                    ;
finishwriteA:       $\{true\}$
                    with $control$ do
finishwriteB:          $\{aw \leq 1\}$
                       $\begin{pmatrix} aw \\ ww \end{pmatrix} := \begin{pmatrix} aw - 1 \\ ww - 1 \end{pmatrix}$
finishwriteC:          $\{aw = 0\}$
                    endwith
                 endwhile
```

$$I_{control} \;=\; \{aw \leq 1\}$$

Fig. 2: Program Readers-Writers and its smallest Owicki-Gries-core proof.

The strongest Owicki-Gries-core-provable property is

$$\begin{aligned}
\{v \in \mathit{ConsState} \mid\ & v(ww), v(ar) \in \mathbb{N}_0 \ \wedge\ v(aw) \in \{0,1\} \\
& \wedge \forall\, i \in \{1,\ldots,n\} : \\
&\qquad (v(pc_{\mathtt{reader}_i}) \in \{x\mathtt{ready}y, \mathtt{read} \mid x \in \{\mathtt{start},\mathtt{finish}\} \\
&\qquad\quad \wedge\, y \in \{\mathtt{A},\mathtt{B},\mathtt{C}\}\}) \\
&\qquad \wedge (v(pc_{\mathtt{reader}_i}) = \mathtt{startreadB} \Rightarrow v(ww) = 0) \\
&\qquad \wedge (v(pc_{\mathtt{reader}_i}) = \mathtt{startreadC} \Rightarrow v(ww) = 0 < v(ar)) \\
& \wedge \forall\, j \in \{1,\ldots,m\} : \\
&\qquad (v(pc_{\mathtt{writer}_j}) \in \{x\mathtt{write}y, \mathtt{write} \mid x \in \{\mathtt{ask},\mathtt{start},\mathtt{finish}\} \\
&\qquad\quad \wedge\, y \in \{\mathtt{A},\mathtt{B},\mathtt{C}\}\}) \\
&\qquad \wedge (v(pc_{\mathtt{writer}_j}) = \mathtt{askwriteC} \Rightarrow v(ww) > 0) \\
&\qquad \wedge (v(pc_{\mathtt{writer}_j}) = \mathtt{startwriteB} \Rightarrow v(ar) = v(aw) = 0) \\
&\qquad \wedge (v(pc_{\mathtt{writer}_j}) = \mathtt{startwriteC} \Rightarrow v(ar) = 0 < v(aw)) \\
&\qquad \wedge (v(pc_{\mathtt{writer}_j}) = \mathtt{finishwriteC} \Rightarrow v(aw) = 0)\,\}\,.
\end{aligned}$$

There are states in this set in which more than one writer is at `write`. Thus no Owicki-Gries-core proof can show that a writer has exclusive access. The smallest Owicki-Gries-core proof can prove a slightly weaker property, namely that the value of *aw*, which tracks the number of writers, never exceeds 1.

Example 2 (progUpper). The program Upper in Fig. 3 will be used for showing differences between the Owicki-Gries-core proofs and the upper bound later. The range of the data variables is $\{0,1\}$. The computation is trivial: no thread can make a step. The majority of the program text serves to create a particular distribution of variables into locals and resources. There are exactly two reachable states, namely the

```
                    initially $u \neq z = x = y = l$
                    resource  $r(u,z)$,  $r'(x,y)$
// Thread 1                               || // Thread 2
A: $\{l = x\}$                            || O: $\{y = z\}$
   with $r$ when $l = u$ do               ||    assume $false$;
B:    $\{l = x = u \neq z\}$              || P: with $r'$ do
      with $r'$ do                        || Q:    $y := 0$
C:       $\{y \geq l = x = u \neq z\}$    || R: endwith;
         $x := 0$                         || S: with $r$ do
D:       $\{y \geq l = u \neq z \wedge x = 0\}$ || T:    $u := 0$;
      endwith;                            || U:    $z := 0$
E:    $\{l = u \neq z \wedge x = 0\}$     || V: endwith;
      assume $false$                      || W:
F: endwith;                               ||
G: with $r'$ do                           ||
H:    $x := 0$                            ||
I: endwith;                               ||
J: with $r$ do                            ||
K:    $u := 0$                            ||
L: endwith;                               ||
M: $l := 0$                               ||
N:                                        ||
                    $I_r = \{u \neq z\}$,   $I_{r'} = \{x \leq y\}$
```

Fig. 3: Program Upper and its smallest Owicki-Gries-core proof. Control flow locations following "`assume` *false*" are annotated by $\{false\}$ by default.

initial ones. The smallest Owicki-Gries-core proof denotes many more states, e.g., $[u \mapsto 0, z \mapsto 1, x \mapsto 0, y \mapsto 1, l \mapsto 0, pc_1 \mapsto \mathtt{A}, pc_2 \mapsto \mathtt{O}]$.

Example 3 (Simple). A program belongs to the class *Simple* if it has at most one resource, and if the resource exists, then it contains no local variables and all its proof variables belong to the resource. Readers-Writers is a family of programs from Simple.

Example 4 (SepThreads). The class *SepThreads* (Separate Threads) consists of all programs that have no resources and whose initial states are exactly those that satisfy the initial conditions of all the threads (a proper subset of such states is disallowed). For such programs the smallest Owicki-Gries-core proof denotes the set of reachable states.

4 Owicki-Gries core as abstract interpretation

4.1 Owicki-Gries-core proofs are the post-fixpoints of a sound abstract successor map

Our first step to try to give a measure of the precision of the Owicki-Gries core will be to cast it in the abstract interpretation framework. An Owicki-Gries-core proof is clearly an abstraction of the set of reachable states of a program. So we will give a characterization of Owicki-Gries core as a post-fixpoint of a sound abstraction of the successor map *post* of an RPL program. The map mimics the application of the Owicki-Gries-core proof conditions.

For a program annotation (I,M), we define the *sound Owicki-Gries-core abstract successor map* $post^{\#}_{\mathrm{OG}}(I,M) = (I',M')$, which applies the sequential consistency once:

- I'_r is the smallest superset of I_r that contains all valuations w of proof variables of r such that there is a transition $v \rightarrow_t v'$ (for some thread t) that exits a critical section for r, v satisfies $M_{v(pc_t),t}$ and w equals the projection of v' on the proof variables of r;
- $M'_{p',t}$ is the smallest superset of $M_{p',t}$ that contains all valuations w of local data variables of thread t and proof variables of resources held at location p' such that there is a transition $v \rightarrow_t v'$ such that v satisfies $M_{v(pc_t),t}$, v satisfies I_r if the transition starts a critical section for r, $v'(pc_t)=p'$, and w equals the projection of v' to local data variables of t and to the proof variables of resources held at p'.

This operator is, as the name says, sound with respect to the successor map.
Let (I^{init},M^{init}) be the annotation describing the initial states only:

- I^{init}_r is the set of all valuations of proof variables of r in the initial states;
- $M^{init}_{p,t}$ is the set of valuations of local data variables of thread t in the initial states.

The *Owicki-Gries-core abstract transformer* $F^{\#}_{\mathrm{OG}}(I,M)$ is constructed as the pointwise union of (I^{init},M^{init}) and $post^{\#}_{\mathrm{OG}}(I,M)$. The set of post-fixpoints of $F^{\#}_{\mathrm{OG}}$ coincides with the set of Owicki-Gries-core proofs; thus the following theorem holds.

Theorem 5 (Equivalence). *The least fixpoint of $F^{\#}_{\mathrm{OG}}$ is the smallest Owicki-Gries-core proof.*

4.2 Characteristic closures for Owicki-Gries core?

An elegant way of describing the precision of an approximation of a semantics given in fixpoint form is through the use of *closures*. A closure ρ is a monotone operator that is idempotent ($\rho(\rho(x)) = \rho(x)$) and extensive ($\rho(x) \geq x$). Given a concrete domain $(D,\leq)$ and a monotone function $F : D \rightarrow D$, the closure ρ on D defines an approximation of the concrete semantics $lfp(F)$ in the sense that $lfp(F) \leq lfp(\rho \circ F)$. Such a description shows the actual loss of information, as the fixpoints of the closure are exactly the abstract elements that describe the approximation, and applying the closure to one concrete element exactly shows what information is lost for that element.

In our case, we have a concrete domain of sets of consistent states ordered by inclusion, and the semantics is given as the least fixpoint of the successor map *post* over the initial states. Then we exhibited an approximation $post^{\#}_{\mathrm{OG}}$ of that successor map. In order to describe its precision, it would be nice to find a closure ρ such that $post^{\#}_{\mathrm{OG}}$ is exactly the best transformer for $\rho \circ post$. Finding one closure for each program is not difficult (just take the closure with two fixpoints, one being the strongest Owicki-Gries-core-provable property and the other the set of all consistent states), but this would not be very informative. Instead, because the concrete domain is entirely fixed by the syntactic structure of the program, we would like to find a ρ that would be fixed for a given syntactic structure. Alas, the next section shows that it is not possible in general.

5 Absence of equivalent characterizations by closures

The main result of this section is unfortunately negative: assuming we start from a reachable state semantics, there is no way to describe the strongest Owicki-Gries-core proof for a given syntactic structure using closures.

Theorem 6 (No Equivalence). *There is a syntactic structure such that for consistent states defined through the syntactic structure and the concrete domain being the powerset of consistent states, there is no closure ρ on the powerset of consistent states such that for any multithreaded program having the given syntactic structure and for any property S of consistent states we have*

$$S \text{ is Owicki-Gries-core provable} \quad \Leftrightarrow \quad lfp(\lambda x.\, \rho(init \cup post(x))) \subseteq S.$$

One such syntactic structure is given by program Upper from Example 2.

In fact, we can prove an even stronger property, as the proof requires only that ρ is monotone. Even more, the same proof holds even if we restrict the validity of the equivalence to a given transition relation:

Theorem 7. *Under the same syntactic structure as in Thm. 6, there is no family of monotone maps Ξ indexed by transition relations, such that Ξ preserves least fixpoints ($\forall$ transition relations τ_1, τ_2: $lfp(\tau_1) = lfp(\tau_2) \Rightarrow lfp(\Xi_{\tau_1}) = lfp(\Xi_{\tau_2})$), and for any property S of consistent states we have*

$$S \text{ is Owicki-Gries-core provable} \quad \Leftrightarrow \quad lfp(\Xi_{\lambda x. init \cup post(x)}) \subseteq S.$$

Theorem 6 shows that if we start by approximating traces by states and wish to describe the Owicki-Gries-core proof system using closures, we can only hope for bounds framing the proof system. We will provide such bounds in the next two sections. If we are willing to work with closures on sets of traces instead of sets of states, it might be possible to find some equivalence. But such an equivalence cannot be obtained directly by collecting the states of traces obtained from abstract interpretation with a closure. Let $\widetilde{\alpha}$ be the abstraction which associates to a set of traces the set of consistent states appearing in the traces.

Theorem 8. *Under the same syntactic structure as in Thm. 6, there is no monotone operator* $\widetilde{\rho}$ *on the powerset of traces of consistent states such that for any multithreaded program having the given syntactic structure, set of initial traces* $\widetilde{init}$ *and the trace extension operator* $\widetilde{post}$ *and for any property S of consistent states we have*

$$S \text{ is Owicki-Gries-core provable} \quad \Leftrightarrow \quad \widetilde{\alpha}(lfp(\lambda x.\, \widetilde{\rho}(\widetilde{init} \cup \widetilde{post}(x)))) \subseteq S.$$

6 Upper bound on precision

Now we will show a closure operator such that the best abstract interpretation of the program with this approximation allows proving a larger set of properties than those provable by the Owicki-Gries core. The approximation will depend only on the syntactic structure, but not on the exact transition relation of a program.

Definition 9 (Owicki-Gries-core annotation closure). *For a given syntactic structure, let* $\underline{\rho}(Q)$ *be the approximation defined as the set of consistent states v such that:*

- *if a resource r is available in v, then there is some other state in Q*
  - *that coincides with v on the proof variables of r and*
  - *in which r is available;*
- *and for any thread t there is a state in Q that coincides with v on local variables (including the program counter) of t and on the proof variables of the resources held by t in v.*

Then $\underline{\rho}$ *is a closure on the powerset of consistent states. We call it* the Owicki-Gries-core annotation closure.

The reason why we call this closure an Owicki-Gries-core annotation closure is because it is the closure corresponding to the Galois connection defined by γ_{OG} (which gives the set of consistent states denoted by a program annotation).

Now fix an arbitrary program and let $\underline{\rho}$ be defined by its syntactic structure.

Theorem 10 (Upper bound). *Abstract interpretation with* $\underline{\rho}$ *is at least as strong as the Owicki-Gries core. Formally:*

$$lfp(\lambda x.\, \underline{\rho}(init \cup post(x))) \;\subseteq\; \text{the strongest Owicki-Gries-core-provable property}.$$

From the high-level view, the best transformer using this closure is capable of taking into account *global* computation instead of *local* successor computation in the Owicki-Gries core. It is as if before checking sequential consistency we take into account annotations not only of one thread, but of all the threads, gaining precision.

Furthermore, the Owicki-Gries-core annotation closure shows where the information is always lost. For instance, if a syntactic structure says that locals are disjoint among themselves and from all the proof variables of the resources, and if two states outside of critical sections are given, then any combination of the locals and resource variables of those states is in the approximation of those two states.

Example 11 (Readers-Writers). For the program Readers-Writers from Example 1 the least fixpoint of $lfp(\lambda x.\, \underline{\rho}(init \cup post(x)))$ coincides with the strongest Owicki-Gries-core-provable property. This property is coarser than the set of reachable states. For example, in one execution $\mathtt{writer}_1$ can reach `write`, in another execution $\mathtt{writer}_2$ can reach `write`, and the initial state satisfies $ww = ar = aw = 0$, so $\underline{\rho}$ produces a combined state where both writers are at `write`, other threads are at their initial locations and all data variables have value 0. Thus, no Owicki-Gries-core proof can restore the dependency between the threads and prove mutual exclusion between the writers.

Example 12 (Upper). For the program Upper from Example 2 abstract interpretation with $\underline{\rho}$ produces the set of reachable states, which is properly included into every Owicki-Gries-core-provable property. The reason for this discrepancy is that locals of one thread and resources held by a different thread overlap. Such an overlap constrains the output of $\underline{\rho}$, but not an Owicki-Gries-core proof. Considering such overlaps actually improves precision!

Example 13 (Simple). For the programs of the class Simple abstract interpretation with $\underline{\rho}$ produces the same result as the smallest Owicki-Gries-core proof. The reason, as we will see, is that abstract interpretation with the lower bound will produce the same result as abstract interpretation with the upper bound.

Example 14 (SepThreads). For the programs of the class SepThreads of Example 4 abstract interpretation with $\underline{\rho}$ produces the same result as the smallest Owicki-Gries-core proof. The reason is that the smallest Owicki-Gries-core proof already denotes the set of reachable states.

7 Lower bound on precision

Now we will show a nontrivial Cartesian-like closure such that abstract interpretation with this closure can prove only properties weaker than or equal to the strongest Owicki-Gries-core-provable property. In fact, we will even show two such closures. One closure (namely $\overline{\rho}$) will describe a better lower bound, while the other one (namely ρ_{c}) is easier to comprehend.

The definitions of both bounds require some preparation.

For a family of sets $\mathscr{X} = \{X_1, \dots, X_n\}$, let $Part(\mathscr{X})$, called *partition* of $\mathscr{X}$, be the set of all nonempty $Y_1 \cap \dots \cap Y_n$ where each Y_i is either X_i or its complement $(\bigcup \mathscr{X}) \setminus X_i$ ($1 \le i \le n$). Elements of a partition are called *blocks*.

Let us fix an arbitrary syntactic structure. Our lower bounds will depend on the syntactic structure but be the same for all the programs that obey this syntactic structure. Let RL be the family of sets containing all resources r and all sets of locals $Local_t$ for all threads t as elements. Let $\widetilde{RL}$ be the partition of RL.

For example, for the syntactic structure of Upper the set RL has exactly four elements: the locals of the first thread $\{l, x, pc_1\}$, the locals of the second thread $\{y, z, pc_2\}$, the resource $r = \{u, z\}$, and the resource $r' = \{x, y\}$. The corresponding partition has exactly six blocks: $\{u\}$, $\{x\}$, $\{y\}$, $\{z\}$, $\{l, pc_1\}$, $\{pc_2\}$.

7.1 Simple cartesian closure

The simpler approximation is defined as follows.

Definition 15 (Cartesian closure). *Given a set of consistent states Q, the* Cartesian *approximation ρ_c returns all the consistent states from the product of projections of Q onto the blocks in $\widetilde{RL}$.*
In other words, $\rho_c(Q)$ contains exactly those consistent states v such that for each block there is a state $\tilde{v}$ in Q that agrees with v on the variables of the block.

As the name says, ρ_c is a closure.

This approximation breaks all the dependencies between the blocks and retains all dependencies inside a block.

For example, for the syntactic structure of the program Upper the variables l and pc_1 belong to the same block. Thus, if in a set of states every state at some fixed control flow location satisfies $l = 0$, each state from the Cartesian approximation of this set will also satisfy $l = 0$ for that control flow location.

Abstract interpretation with ρ_c generates a property which is always weaker than or equal to the strongest Owicki-Gries-core-provable property.

7.2 More precise lower bound closure

The following definition strengthens ρ_c in two ways. Firstly, we impose restrictions on $\tilde{v}$ from the definition of ρ_c. Such restrictions will depend on the block. Secondly, we look at the prophecy variables: those are variables which are never written and which don't belong to a resource. Prophecy variables form a separate block; now we restore all the dependencies between this block and those locals of any thread that are not resource variables.

Definition 16 (Lower Bound closure). *Given a set of consistent states Q, its approximation $\overline{\rho}(Q)$ contains exactly those consistent states v such that both of the following conditions are satisfied.*

- *For every block in $\widetilde{RL}$ that is contained in a resource,*
  - *if the resource is available in v, then there is some $\tilde{v} \in Q$*
    - *in which the resource is also available*
    - *and which coincides with v on the block;*
  - *if some thread holds the resource in v, then there is some $\tilde{v} \in Q$ which coincides with v on all the variables*
    - *that are local to this thread but do not belong to any resource, or*
    - *that belong to the block.*
- *For every thread there is a state in Q that coincides with v on each variable*
  - *that is a local variable of the thread but*
  - *does not belong to any resource.*

As the name says, $\overline{\rho}$ is a closure. It is at least as precise as ρ_c. Both closures do not depend on the exact transition relation of a program. The closure $\overline{\rho}$ induces the tightest lower bound we could prove. It shows the dependencies that are always retained, creating a basis for the construction of future refinement algorithms (possibly following [11]).

Now fix an arbitrary RPL program that has the assumed syntactic structure.

The proof of the lower bound relies on several claims about the strongest property provable by abstract interpretation with $\overline{\rho}$. The following claim is the most important one.

Lemma 17. *Let Q be the strongest property provable by abstract interpretation with $\overline{\rho}$. Consider a transition of a thread t from a consistent state v to a consistent state v', let the transition start a critical section. Let $\tilde{v} \in Q$ such that v agrees with $\tilde{v}$ on the locals of t and on the variables of the resources held by t before the transition. Let $\hat{v} \in Q$ such that v agrees with $\hat{v}$ on the variables of the resource being acquired. Then there is a state in Q that agrees with v' on the locals of thread t and on the variables of the resources held by the thread after the transition.*

The lower bound theorem follows from the lemma.

Theorem 18 (Lower bound). *The core of Owicki-Gries can prove at least as many properties as abstract interpretation with $\overline{\rho}$ or ρ_c. Formally:*
the strongest Owicki-Gries-core–provable property $\subseteq$ $\mathit{lfp}(\lambda x.\, \overline{\rho}(\mathit{init} \cup \mathit{post}(x)))$
$\subseteq$ $\mathit{lfp}(\lambda x.\, \rho_c(\mathit{init} \cup \mathit{post}(x)))$.

Example 19 (Readers-Writers). For the program Readers-Writers from Example 1 abstract interpretation with ρ_c produces the set of all consistent states where readers are at ...`read`..., writers are at ...`write`..., and *ww*, *ar*, *aw* are nonnegative. The Owicki-Gries core and abstract interpretation with $\overline{\rho}$ can prove stronger properties, e.g., that at location `askwriteC` the value of *ww* is positive. Intuitively, when a resource is busy, Cartesian abstraction always breaks the dependency between the resource variables and the control flow, while $\overline{\rho}$ and the Owicki-Gries core sometimes retain the dependency.

Example 20 (Upper). For the program Upper from Example 2 abstract interpretation using either ρ_c or $\overline{\rho}$ produces the same result: the set of all states such that the first thread is at any location between `A` and `E`, the second thread is at `0` and all data variables take arbitrary values from $\{0,1\}$. The Owicki-Gries core can prove stronger properties; for instance, it can show that at location `E` the value of x is zero. Intuitively, the dependency between resource variables and control flow is always broken in Cartesian abstraction but is sometimes retained in the Owicki-Gries core.

Example 21 (Simple). For the programs from the class Simple from Example 3 the approximations $\overline{\rho}$ and $\underline{\rho}$ are so close to each other that abstract interpretations with both produce the same property. Since they define the lower and upper bounds on the precision of the Owicki-Gries core, the Owicki-Gries core can prove exactly the same properties as those provable by abstract interpretation with any of the two approximations. If a program from Simple does not have the empty resource, then $\overline{\rho}$ and $\underline{\rho}$ coincide exactly, approximating a set of states Q by the set of all consistent states v such that both of the following conditions hold.

– If there is a resource and it is available in v, there is some state in Q that coincides with v on the resource and in which the resource is available.

– For each thread there is a state in Q which coincides with v on the variables of the thread and, if the resource is present and is held, on the variables of the resource.

Example 22 (SepThreads). For the programs from the class SepThreads from Example 4 the approximations $\bar{\rho}$ and $\underline{\rho}$ coincide. Since they define the lower and upper bounds on the precision of the Owicki-Gries core, the Owicki-Gries core can prove exactly the same properties as those provable by abstract interpretation with any of the two approximations. Due to the absence of any thread interactions and independence of initial states of the threads, the mentioned methods can prove the strongest inductive property, namely the set of states reachable from the initial ones. Informally spoken, all dependencies between the locals of each thread are retained.

8 Related work

Historically, conditional critical regions were introduced in [8]. The thesis [15] of Owicki and her paper with Gries [16] describe the original proof method for RPL. Modular reasoning about RPL has not been characterized in terms of abstract interpretation via closures [5] so far.

For general multithreaded programs (i.e. without separation of data into resources), Owicki-Gries-style reasoning without auxiliary variables is equivalent to multithreaded Cartesian abstraction. The result was first mentioned without proof in [6], and the proof appears in [11].

Clarke [3] has considered a subset of integer RPL programs with only one resource, where, roughly, only additions of constants inside the critical sections are allowed, and the property to be checked is either mutual exclusion of PV-semaphores or deadlock freedom. With a predefined choice of integer auxiliary variables, the least fixpoint of a particular functional is a resource invariant that precisely tells whether the property holds or not. Two overapproximations are given: a fixed-formula resource invariant and an invariant computed by a polyhedral analysis with widening. Our work, on the contrary, does not impose any restrictions on the program form. Our results hold for even more programs than the RPL ones, e.g., where the critical sections are not well-nested.

The work of Owicki on RPL is the basis of a variety of modular programming languages equipped with proof methods of different degrees of completeness and automation.

Concurrent separation logic (CSL) [14] equips RPL with separation logic as a formula language. CSL is also incomplete without the rule of auxiliary variables, so the question of precision arises. Removing secondly important features of CSL for the sake of clarity (as in [2]) and considering variables in the heap makes our lower bound also apply to such CSL versions.

Chalice [10] is a language for verification of object-oriented concurrent programs with heap, equipped with an RPL-like proof system. Due to the powerful permission system, the proof system is in general stronger than that of Owicki, so our lower bound on precision carries over to Chalice for programs that can be directly represented both in RPL and in Chalice.

VCC [13] is a verifier for multithreaded C. When accessing structures in a lock-based manner, VCC requires the user to provide invariants of C `struct`s. On obtaining ownership of a `struct`ure, the resource invariant is assumed; on relinquishing ownership, the resource invariant has to be reestablished. Ghost contracts of lock-manipulating functions control the ownership transfer. Our lower bound applies to VCC as well.

9 Discussion

9.1 Challenges

Discussing the precision loss reveals several open problems.

Example 20 shows a gap between the accuracy of the Owicki-Gries core and the lower bound. Can the lower bound closure be strengthened?

The main inequivalence result assumes that the concrete domain is the powerset of consistent states. For the powerset of traces, we only know inequivalence for a subclass of abstract interpretations. Is there an exact characterization of the Owicki-Gries core by abstract interpretation on the powerset of traces?

We have shown what variable dependencies does the Owicki-Gries core break. Can these dependencies be restored on demand? Is there an automatic counterexample-guided abstraction refinement of the Owicki-Gries core, perhaps based on auxiliary variables [4], unions of Cartesian products [11], or abstract threads [9]?

Can one characterize the precision loss of the Owicki-Gries core by completeness notions of [7]?

Can one formalize CSL in abstract interpretation in a way that would reveal the involved approximation?

9.2 Conclusion

We have examined a modular method (Owicki-Gries core) for proving safety properties of a widely-used class of multithreaded programs.

The considered class contains structured programs in which shared data are partitioned into resources and are accessed only in critical sections that ensure mutually exclusive access to resources. The method provides a clean basis for other more sophisticated proof methods like Concurrent Separation Logic, Chalice, or VCC. The Owicki-Gries core is polynomial in the number of threads, but without manually adding auxiliary variables it cannot prove many properties of concurrent programs.

The Owicki-Gries core is, intuitively, expected to succeed for properties whose dependence on thread coupling is low, and is expected to fail if complicated thread interactions have to be analyzed. We have made this notion precise, providing a characterization of the set of Owicki-Gries-core-provable properties. We have demonstrated an abstract transformer corresponding to the Owicki-Gries core: the least fixpoint of the abstract transformer denotes exactly the strongest Owicki-Gries-core-provable property. To quantify the loss of precision inherent to modularity, we have provided a superset and a subset of Owicki-Gries-core-provable properties, described by abstract interpretations with closure operators that depend on the syntactic structure of the program only.

These bounds coincide for a class of simple programs. We have also shown a principal inability to provide an exact characterization of the set of properties in terms of closures that depend only on the syntactic structure.

10 Acknowledgements

We thank Byron Cook and Microsoft Research Cambridge for support in the initial stages of the work. The work was also partially supported by the Verisoft project of the German science foundation DFG. We thank Josh Berdine and Viktor Vafeiadis for helpful comments and discussions.

References

1. G. Barthe and M. V. Hermenegildo, editors. *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*, volume 5944 of *LNCS*. Springer, 2010.
2. C. Calcagno, P. W. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS*, pages 366–378. IEEE Computer Society, 2007.
3. E. M. Clarke. Synthesis of resource invariants for concurrent programs. *ACM Trans. Program. Lang. Syst.*, 2(3):338–358, 1980.
4. A. Cohen and K. S. Namjoshi. Local proofs for global safety properties. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS 4590*, pages 55–67. Springer, 2007.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
6. R. Cousot. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles*. PhD thesis, Institut national polytechnique de Lorraine, 1985.
7. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *Journal of the ACM*, 47(2):361–416, 2000.
8. C. A. R. Hoare. Towards a theory of parallel programming. In C. A. R. Hoare and R. H. Perrott, editors, *Operating System Techniques*, pages 61–71. Academic Press, 1972.
9. S. K. Lahiri, A. Malkis, and S. Qadeer. Abstract threads. In Barthe and Hermenegildo [1], pages 231–246.
10. K. R. M. Leino. Verifying concurrent programs with Chalice. In Barthe and Hermenegildo [1], page 2.
11. A. Malkis. *Cartesian Abstraction and Verification of Multithreaded Programs*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Feb. 2010.
12. A. Malkis and L. Mauborgne. On the strength of Owicki-Gries for resources. Technical report, IMDEA Software Institute, 2011. `http://software.imdea.org/~alexmalkis/onTheStrengthOfOwickiGriesForResources_techrep.ps`.
13. M. Moskal, W. Schulte, E. Cohen, M. A. Hillebrand, and S. Tobies. Verifying C programs: A VCC tutorial, 2011. MSR Redmond, EMIC Aachen.
14. P. W. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1-3):271–307, 2007.
15. S. S. Owicki. *Axiomatic Proof Techniques For Parallel Programs*. PhD thesis, Cornell University, Department of Computer Science, TR 75-251, July 1975.
16. S. S. Owicki and D. Gries. Verifying properties of parallel programs: an axiomatic approach. *Commun. ACM*, 19(5):279–285, 1976.