# Biabduction (and Related Problems) in Array Separation Logic

James Brotherston[1] and Nikos Gorogiannis[2] and Max Kanovich[1,3]

[1] University College London, UK
[2] Middlesex University, UK
[3] National Research University Higher School of Economics, Russian Federation

**Abstract.** We investigate *array separation logic* (ASL), a variant of symbolic-heap separation logic in which the data structures are either pointers or *arrays*, i.e., contiguous blocks of memory. This logic provides a language for compositional memory safety proofs of array programs. We focus on the *biabduction* problem for this logic, which has been established as the key to automatic specification inference at the industrial scale. We present an NP decision procedure for biabduction in ASL, and we also show that the problem of finding a consistent solution is NP-hard. Along the way, we study satisfiability and entailment in ASL, giving decision procedures and complexity bounds for both problems. We show satisfiability to be NP-complete, and entailment to be decidable with high complexity. The surprising fact that biabduction is simpler than entailment is due to the fact that, as we show, the element of choice over biabduction solutions enables us to dramatically reduce the search space.

**Keywords:** Separation logic, arrays, biabduction, entailment, complexity.

## 1 Introduction

In the last 15 years, *separation logic* [34] has evolved from a novel way to reason about pointers to a mainstream technique for scalable program verification. Facebook's Infer [13] is perhaps the best known tool based on separation logic; other examples include SLAyer [5], VeriFast [28] and HIP [15].

Separation logic is based upon *Hoare triples* of the form $\{A\}\, C\, \{B\}$, where $C$ is a program and $A, B$ are formulas in a logical language. Its compositional nature has two main pillars. The first pillar is the soundness of the *frame rule*:

$$\frac{\{A\}\, C\, \{B\}}{\{A * F\}\, C\, \{B * F\}}\ (\text{Frame})$$

where the *separating conjunction* $*$ is read as "*and separately in memory*", and subject to the restriction that $C$ does not modify any free variables in $F$ [39].

The second pillar is a tractable algorithm for *biabduction* [14]: given formulas $A$ and $B$, find formulas $X$, $Y$ such that $A * X \models B * Y$, usually subject to the proviso that $A * X$ should be satisfiable. Solving this problem enables

us to infer specifications for whole programs given specifications for their individual components [14]. E.g., if $C_1$ and $C_2$ have specifications $\{A'\}\, C_1\, \{A\}$ and $\{B\}\, C_2\, \{B'\}$, we can use a solution $X, Y$ to the above biabduction problem to construct a specification for $C_1; C_2$ as follows, using the frame rule and the usual Hoare logic rules for consequence ($\models$) and sequencing (;):

$$\dfrac{\dfrac{\dfrac{\{A'\}\, C_1\, \{A\}}{\{A' * X\}\, C_1\, \{A * X\}}\,(\text{Frame})}{\{A' * X\}\, C_1\, \{B * Y\}}\,(\models) \qquad \dfrac{\{B\}\, C_2\, \{B'\}}{\{B * Y\}\, C_2\, \{B' * Y\}}\,(\text{Frame})}{\{A' * X\}\, C_1; C_2\, \{B' * Y\}}\,(;)$$

Bottom-up interprocedural analyses based on separation logic, such as Facebook Infer, employ biabduction to infer program specifications from unannotated code. Typically, the underlying assertion language is the "symbolic heap" fragment of separation logic over linked lists [4], which is known to be tractable [16].

Here, we focus on a different, but ubiquitous data structure, namely *arrays*, which we view as contiguous blocks of memory. We propose an *array separation logic* (ASL) in which we replace the usual "list segment" predicate ls by an "array" predicate $\mathsf{array}(a, b)$, which denotes a contiguous block of allocated heap memory from address $a$ to address $b$ (inclusive), as was first proposed in [32]. In addition, since we wish to reason about array bounds, we allow assertions to contain linear arithmetic. Thus, for example, a pointer $x$ to a memory block of length $n > 1$ starting at $a$ can be represented in ASL by the assertion

$$n > 1 : x \mapsto a * \mathsf{array}(a, a + n - 1)\ .$$

The array predicate only records the bounds of memory blocks, not their contents; this is analogous to the abstraction from pointers to lists in standard separation logic. Similar to the situation for lists, memory safety of array-manipulating programs typically depends only on the memory footprint of the arrays.

Our focus is on the biabduction problem for ASL, the most critical step in building a bottom-up memory safety analysis à la Infer for array-manipulating programs. Our first main contribution is a decision procedure for the (quantifier-free) biabduction problem in ASL (Sec. 5). It relies on the idea that, given $A$ and $B$, we can look for some consistent total ordering of all the array endpoints and pointers in both $A$ and $B$, and impose this ordering, which we call a *solution seed*, as the arithmetical part of the solution $X$. Having done this, the computation of the "missing" arrays and pointers in $X, Y$ is a polynomial-time process, and thus the entire algorithm runs in NP-time. We demonstrate that this algorithm is sound and complete, and that the biabduction problem itself is NP-hard, with further bounds for cases involving quantifiers.

We also study the *satisfiability* and *entailment* problems in ASL, and, as our second main contribution, we provide decision procedures and upper/lower complexity bounds for both problems. We find that satisfiability is NP-complete, while entailment is decidable with very high complexity: it can be encoded in $\Pi^0_2$ Presburger arithmetic, and is $\Pi^P_2$-hard. It may at first sight appear surprising

that entailment is harder than biabduction, as biabduction also seems to involve solving an entailment problem. However, in biabduction, there is an element of *choice* over $X, Y$, and we exploit this to dramatically reduce the cost of checking these conditions. Namely, committing to a specific solution seed (see above) reduces biabduction to a simple computation rather than a search problem.

The remainder of this paper is structured as follows. Section 2 gives an example motivating the ASL biabduction problem in practice. The syntax and semantics of ASL is given formally in Section 3. We present algorithms and complexity bounds for satisfiability, biabduction and entailment for ASL in Sections 4, 5 and 6 respectively. Section 7 surveys related work, and Section 8 concludes.

Due to space limitations, the proofs of the results in this paper are omitted or only sketched. They are, however, available in the long version of this article [11].

## 2 Motivating example

Here, we give a simple example illustrating how the biabduction problem arises when verifying array programs, using ASL as our assertion language. Our example is deliberately high-level, in order to illustrate some key features of the general problem. However, more concrete examples, involving concrete array programs, can be found in section 2 of [11].

Suppose we have a procedure `foo` that manipulates an array somehow, with specification $\{\mathsf{array}(c,d)\}\,\mathtt{foo}(\mathtt{c},\mathtt{d})\,\{Q\}$ (supplied in advance, or computed at an earlier stage of the analysis). Now, consider a procedure including a call to `foo`, say $C;\mathtt{foo}(\mathtt{c},\mathtt{d});\ldots$, and suppose that we have computed the specification $\{\mathsf{emp}\}\,C\,\{\mathsf{array}(a,b)\}$, say, for the code $C$ prior to this call. As in the Introduction, this gives rise to the biabduction problem

$$\mathsf{array}(a,b) * X \models \mathsf{array}(c,d) * Y$$

with the effect that $\{X\}\,C;\mathtt{foo}(\mathtt{c},\mathtt{d})\,\{Q * Y\}$ then becomes a valid specification for the initial code including the call to `foo`.

Solving this problem depends crucially on the position in memory of $c$ and $d$ relative to $a$ and $b$; depending on *whether* and *how* the arrays $\mathsf{array}(a,b)$ and $\mathsf{array}(c,d)$ overlap, we have to add different arrays to $X$ and $Y$ so that the memory footprint of the two sides becomes the same. Such ordering information *might* be available as part of the postcondition computed for $C$; if not, then we have to guess it, as part of the "antiframe" $X$. The solutions include:

| | | |
|---|---|---|
| $X := a = c \wedge b = d : \mathsf{emp}$ | and | $Y := \mathsf{emp}$ |
| $X := d < a : \mathsf{array}(c,d)$ | and | $Y := \mathsf{array}(a,b)$ |
| $X := a < c \wedge d < b : \mathsf{emp}$ | and | $Y := \mathsf{array}(a, c-1) * \mathsf{array}(b+1, d)$ |
| $X := a < c < b < d : \mathsf{array}(b+1, d)$ | and | $Y := \mathsf{array}(a, c-1)$ |

*et cetera*. Note that these solutions are all (a) spatially minimal, relative to the ordering constraints in $X$ (i.e. the arrays are as small as possible), and (b) logically incomparable to one another. Thus, when dealing with arrays in separation logic, any complete biabduction algorithm *must* take into account the possible ways in which the arrays might be positioned relative to each other.

## 3 Array separation logic, ASL

Here, we present separation logic for arrays, ASL, which employs a similar *symbolic heap* formula structure to that in [4], but which treats contiguous *arrays* in memory rather than linked list segments; we additionally allow linear arithmetic.

**Definition 3.1 (Symbolic heap).** Terms $t$, pure formulas $\Pi$, spatial formulas $F$ *and* symbolic heaps $SH$ *are given by the following grammar:*

$$\begin{aligned} t &::= x \mid n \mid t + t \mid nt \\ \Pi &::= t = t \mid t \neq t \mid t \leq t \mid t < t \mid \Pi \wedge \Pi \\ F &::= \mathsf{emp} \mid t \mapsto t \mid \mathsf{array}(t, t) \mid F * F \\ SH &::= \exists \mathbf{z}.\ \Pi : F \end{aligned}$$

*where $x$ ranges over an infinite set* Var *of* variables, $\mathbf{z}$ *over sets of variables, and $n$ over $\mathbb{N}$. Whenever one of $\Pi, F$ is empty in a symbolic heap, we omit the colon. We write $FV(A)$ for the set of free variables occurring in $A$. If $A = \exists \mathbf{z}.\ \Pi : F$ then we write $\mathsf{qf}(A)$ for $\Pi : F$, the* quantifier-free *part of $A$.*

We interpret this language in a stack-and-heap model, where both locations and values are natural numbers. A *stack* is a function $s\colon \mathsf{Var} \to \mathbb{N}$. We extend stacks over terms as usual: $s(n) = n$, $s(t_1 + t_2) = s(t_1) + s(t_2)$ and $s(nt) = ns(t)$. If $s$ is a stack, $z \in \mathsf{Var}$ and $m \in \mathbb{N}$, we write $s[z \mapsto v]$ for the stack defined as $s$ except that $s[z \mapsto v](z) = v$. We extend stacks pointwise over term tuples.

A *heap* is a finite partial function $h\colon \mathbb{N} \rightharpoonup_{\text{fin}} \mathbb{N}$ mapping finitely many locations to values; we write $\mathrm{dom}\,(h)$ for the domain of $h$, and $e$ for the empty heap that is undefined on all locations. We write $\circ$ for *composition* of domain-disjoint heaps: if $h_1$ and $h_2$ are heaps, then $h_1 \circ h_2$ is the union of $h_1$ and $h_2$ when $\mathrm{dom}\,(h_1)$ and $\mathrm{dom}\,(h_2)$ are disjoint, and undefined otherwise.

**Definition 3.2.** *The* satisfaction relation $s, h \models A$*, where $s$ is a stack, $h$ a heap and $A$ a symbolic heap, is defined by structural induction on $A$.*

$$\begin{aligned} s,h \models t_1 \sim t_2 &\Leftrightarrow s(t_1) \sim s(t_2) \text{ where } \sim \text{ is } =, \neq, <, \text{ or } \leq \\ s,h \models \Pi_1 \wedge \Pi_2 &\Leftrightarrow s,h \models \Pi_1 \text{ and } s,h \models \Pi_2 \\ s,h \models \mathsf{emp} &\Leftrightarrow h = e \\ s,h \models t_1 \mapsto t_2 &\Leftrightarrow \mathrm{dom}\,(h) = \{s(t_1)\} \text{ and } h(s(t_1)) = s(t_2) \\ s,h \models \mathsf{array}(t_1, t_2) &\Leftrightarrow s(t_1) \leq s(t_2) \text{ and } \mathrm{dom}\,(h) = \{s(t_1), \ldots, s(t_2)\} \\ s,h \models F_1 * F_2 &\Leftrightarrow \exists h_1, h_2.\ h = h_1 \circ h_2 \text{ and } s,h_1 \models F_1 \text{ and } s,h_2 \models F_2 \\ s,h \models \exists \mathbf{z}.\ \Pi : F &\Leftrightarrow \exists \mathbf{m} \in \mathbb{N}^{|\mathbf{z}|}.\ s[\mathbf{z} \mapsto \mathbf{m}], h \models \Pi \text{ and } s[\mathbf{z} \mapsto \mathbf{m}], h \models F \end{aligned}$$

Satisfaction of pure formulas $\Pi$ does not depend on the heap; we write $s \models \Pi$ to mean that $s, h \models \Pi$ (for any heap $h$). We write $A \models B$ to mean that $A$ *entails* $B$, i.e. that $s, h \models A$ implies $s, h \models B$ for all stacks $s$ and heaps $h$.

*Remark 3.3.* Our array predicate employs *absolute* addressing: $\mathsf{array}(k, \ell)$ denotes an array from $k$ to $\ell$. In practice, one often reasons about arrays using *base-offset* addressing, where $\mathsf{array}(b, i, j)$ denotes an array from $b + i$ to $b + j$. We

can define such a ternary version of our array predicate, overloading notation, by $\mathsf{array}(b, i, j) =_{\mathrm{def}} \mathsf{array}(b{+}i, b{+}j)$. Conversely, any $\mathsf{array}(k, \ell)$ can be represented in base-offset style as $\mathsf{array}(0, k, \ell)$. Thus, we may freely switch between absolute and base-offset addressing.

Satisfiability in the unrestricted pure part of our language is already NP-hard. Thus, in order to obtain sharper complexity results, we will sometimes confine our attention to symbolic heaps in the following special *two-variable form*.

**Definition 3.4.** *A symbolic heap* $\exists \mathbf{z}.\ \Pi : F$ *is said to be in* two-variable form *if*

*(a) its pure part* $\Pi$ *is a conjunction of* 'difference constraints' *of the form* $x = k$, $x = y + k$, $x \leq y + k$, $x \geq y + k$, $x < y + k$, *and* $x > y + k$, *where* $x$ *and* $y$ *are variables, and* $k \in \mathbb{N}$*; (notice that* $x \neq y$ *is not here);*
*(b) its spatial part* $F$ *contains only formulas of the form* $k \mapsto v$, $\mathsf{array}(a, 0, j)$, $\mathsf{array}(a, 1, j)$, *and* $\mathsf{array}(k, j, j)$, *where* $v$, $a$, *and* $j$ *are variables, and* $k \in \mathbb{N}$.

When pure formulas are conjunctions of 'difference constraints' as in Definition 3.4, their satisfiability becomes polynomial [17].

## 4 Satisfiability in ASL

Here, we show that *satisfiability* in ASL is NP-complete. This stands in contrast to the situation for symbolic-heaps over list segments, where satisfiability is polynomial [16], and over general inductive predicates, where it is EXP-complete [10].

**Satisfiability problem for ASL.** *Given symbolic heap* $A$*, decide if there is a stack* $s$ *and heap* $h$ *with* $s, h \models A$.

First, we show that satisfiability of a symbolic heap can be encoded as a $\Sigma^0_1$ formula of *Presburger arithmetic* and can therefore be decided in NP time.

**Definition 4.1.** Presburger arithmetic (PbA) *is defined as the first-order theory (with equality) of the natural numbers* $\mathbb{N}$ *over the signature* $\langle 0, s, + \rangle$*, where* $s$ *is the successor function, and* $0$ *and* $+$ *have their usual interpretations. It is immediate that the relations* $\neq$, $\leq$ *and* $<$ *can be encoded (possibly introducing an existential quantifier), as can the operation of multiplication by a constant.*

Note that a stack is just a standard first-order valuation, and that a pure formula in ASL is also a formula of PbA. Moreover, the satisfaction relations for ASL and PbA coincide on such formulas. Thus, we overload $\models$ to include the standard first-order satisfaction relation of PbA.

The intuition behind our encoding of satisfiability is that a symbolic heap is satisfiable exactly when the pure part is satisfiable, each array is well-defined, and all pointers and arrays are non-overlapping with all of the others. For simplicity of exposition, we do this by abstracting away pointers with single-cell arrays.

**Definition 4.2.** *Let $A$ be a quantifier-free symbolic heap, written (without loss of generality) in the form: $A = \Pi : \circledast_{i=1}^{n} \mathsf{array}(a_i, b_i) * \circledast_{i=1}^{m} c_i \mapsto d_i$. We define its* array abstraction *as*

$$\lfloor A \rfloor =_{def} \Pi : \circledast_{i=1}^{n} \mathsf{array}(a_i, b_i) * \circledast_{i=1}^{m} \mathsf{array}(c_i, c_i) \ .$$

**Lemma 4.3.** *Let $A$ be a quantifier-free symbolic heap and $s$ a stack. Then, $\exists h.\, s, h \models A$ iff $\exists h'.\, s, h' \models \lfloor A \rfloor$.*

**Definition 4.4.** *Let $A$ be a quantifier-free symbolic heap, and let $\lfloor A \rfloor$ be of the form $\Pi : \circledast_{i=1}^{n} \mathsf{array}(a_i, b_i)$. We define a corresponding formula $\gamma(A)$ of* PbA *as*

$$\gamma(A) =_{def} \Pi \wedge \bigwedge_{1 \leq i \leq n} a_i \leq b_i \wedge \bigwedge_{1 \leq i < j \leq n} (b_i < a_j) \vee (b_j < a_i) \ .$$

Note that $\gamma(A)$ is defined in terms of the abstraction $\lfloor A \rfloor$.

**Lemma 4.5.** *For any stack $s$ and any quantifier-free symbolic heap $A$, we have $s \models \gamma(A)$ iff $\exists h.\ s, h \models A$.*

**Proposition 4.6.** *Satisfiability for* ASL *is in* NP.

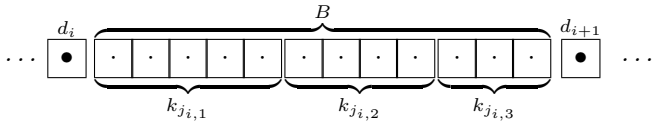
*Proof.* Follows from Lemma 4.5 and the fact that satisfiability for $\Sigma_1^0$ Presburger arithmetic is in NP [35]. □

Prop. 4.6 may also be obtained by viewing ASL as a sub-fragment of the *array property fragment* [8]. However, we put forward Defn. 4.4 and Lemma 4.5 as we make heavy use of them in Sec. 5.

Satisfiability is shown NP-hard by reduction from the 3-*partition problem [21].*

**3-partition problem.** *Given $B \in \mathbb{N}$ and a sequence of natural numbers $\mathcal{S} = (k_1, k_2, \ldots, k_{3m})$ such that $\sum_{j=1}^{3m} k_j = mB$, and $B/4 < k_j < B/2$ for all $j \in [1, 3m]$, decide whether there is a partition of $\mathcal{S}$ into $m$ groups of three, say $\{(k_{j_{i,1}}, k_{j_{i,2}}, k_{j_{i,3}}) \mid i \in [1, m]\}$, such that $k_{j_{i,1}} + k_{j_{i,2}} + k_{j_{i,3}} = B$ for all $i \in [1, m]$.*

**Definition 4.7.** *Given an instance $(B, \mathcal{S})$ of the 3-partition problem, we define a symbolic heap $A_{B,\mathcal{S}}$ as follows. First we introduce $m + 1$ numbers $d_i$ acting as single-cell "delimiters" between chunks of memory of length $B$, (therefore, $d_{i+1} = d_i + B + 1$), and $a_j$ to allocate arrays of length $k_j$ in the space between some pair of delimiters $d_i$ and $d_{i+1}$. Visually, the arrangement is as follows:*




*Concretely, $A_{B,\mathcal{S}}$ is the following symbolic heap:*

$$\bigwedge_{j=1}^{3m} (d_1 \leq a_j) \wedge (a_j + k_j < d_{m+1}) : \circledast_{i=1}^{m+1} \mathsf{array}(d_i, 0, 0) * \circledast_{j=1}^{3m} \mathsf{array}(a_j, 1, k_j) \ .$$

**Lemma 4.8.** *Given a 3-partition problem $(B, \mathcal{S})$, and letting $A_{B,\mathcal{S}}$ be given by Defn. 4.7, we have that $A_{B,\mathcal{S}}$ is satisfiable iff there is a 3-partition of $\mathcal{S}$ (w.r.t. $B$).*

**Theorem 4.9.** *The satisfiability problem for* ASL *is* NP*-complete, even when symbolic heaps are restricted to be quantifier-free, and in two-variable form.*

*Proof.* Prop. 4.6 provides the upper bound. For the lower bound, Defn. 4.7 and Lemma 4.8 establish a polynomial reduction from the 3-partition problem. □

## 5 Biabduction

Here, we turn to the central focus of this paper, *biabduction* for ASL. In stating this problem, it is convenient to first lift the connective $*$ to symbolic heaps:

$$(\exists \mathbf{x}.\ \Pi : F) * (\exists \mathbf{y}.\ \Pi' : F') = \exists \mathbf{x} \cup \mathbf{y}.\ \Pi \wedge \Pi' : F * F' \ ,$$

where the existentially quantified variables $\mathbf{x}$ and $\mathbf{y}$ are assumed disjoint, and no free variable capture occurs (this can always be avoided by $\alpha$-renaming).

**Biabduction problem for** ASL**.** *Given satisfiable symbolic heaps $A, B$, find symbolic heaps $X, Y$ such that $A * X$ is satisfiable and $A * X \models B * Y$.*

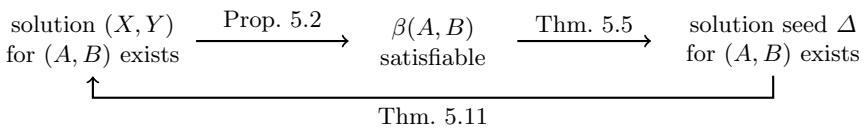
We first consider quantifier-free biabduction (Sec. 5.1), and investigate its complexity in Sec. 5.2. We then show that when quantifiers appear in $B, Y$ which are appropriately restricted, existence of solutions can be decided using the machinery for the quantifier-free case (Sec. 5.3). In the same section we also characterise the complexity of biabduction in the presence of quantifiers.

### 5.1 An algorithm for quantifier-free biabduction

We give an algorithm for quantifier-free biabduction. Let $(A, B)$ be a biabduction problem and $(X, Y)$ a solution. The intuition is that a model $s, h$ of both $A$ *and* $B$ induces a total order over the terms of $A, B$, dictating the form of $X, Y$.

Consider Fig. 1, which depicts a biabduction instance $(A, B)$ and a solution $(X, Y)$, where all array endpoints in $A, B$ are totally ordered. Using this order, we can compute $X, Y$ by covering parts that $B$ requires but $A$ does not provide $(X)$ and by covering parts that $A$ requires but $B$ does not provide $(Y)$.

We capture this intuition by introducing a formula $\Delta$, called a *solution seed*, capturing the total order over the terms of $A, B$. We show that the existence of a solution seed $\Delta$ implies the existence of a solution $(X, Y)$ for the biabduction problem $(A, B)$, and is in turn implied by the satisfiability of a certain PbA formula $\beta(A, B)$. To complete the circle, we show that $\beta(A, B)$ is satisfiable whenever there is a biabduction solution for $(A, B)$:



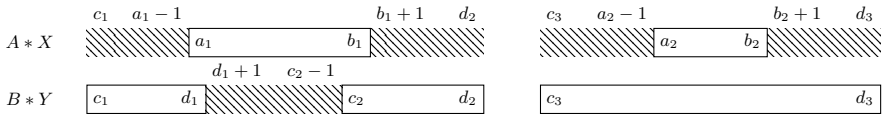



**Fig. 1.** Example showing solutions in Defn. 5.6. Arrays of $A, B$ are displayed as boxes and arrays in $X, Y$ as hatched rectangles.

Finally, we show that the problem of finding a solution to a biabduction problem is in NP and that our algorithm is complexity-optimal (Prop. 5.13).

**Definition 5.1 (The formula $\beta$).** *Let $(A, B)$ be an biabduction instance, where*

$$A = \Pi : \bigstar_{i=1}^{n} \mathsf{array}(a_i, b_i) * \bigstar_{i=1}^{k} t_i \mapsto u_i \qquad B = \Pi' : \bigstar_{i=1}^{m} \mathsf{array}(c_i, d_i) * \bigstar_{i=1}^{\ell} v_i \mapsto w_i$$

*We define a formula $\beta(A, B)$ of* PbA *as follows:*

$$\beta(A,B) =_{def} \gamma(A) \wedge \gamma(B) \wedge \bigwedge_{j=1}^{\ell} \bigwedge_{i=1}^{n} (v_j < a_i \vee v_j > b_i) \wedge \bigwedge_{i=1}^{\ell} \bigwedge_{j=1}^{k} (t_i \neq v_j \vee u_i = w_j)$$

**Proposition 5.2.** *If $(A, B)$ has a solution, then $\beta(A, B)$ is satisfiable.*

*Proof.* (Sketch) Letting $X, Y$ be a solution for $(A, B)$, there is a model $s, h$ of $A * X$. We show that $s \models \beta(A, B)$, using Lemma 4.5 for the first conjunct of $\beta$, and the fact that $A * X \models B * Y$ for the other conjuncts. □

Given an instance of the form in Defn. 5.1, we define a set $\mathcal{T}_{A,B}$ of terms by:

$$\mathcal{T}_{A,B} =_{\text{def}} T(A) \cup T(B) \cup \bigcup_{i=1}^{n} \{b_i + 1\} \cup \bigcup_{i=1}^{m} \{d_i + 1\} \cup \bigcup_{i=1}^{k} \{t_i + 1\} \cup \bigcup_{i=1}^{\ell} \{v_i + 1\}$$

where $T(-)$ denotes the set of all terms in a symbolic heap.

**Definition 5.3 (Solution seed).** *A* solution seed *for a biabduction problem $(A, B)$ in the form of Defn. 5.1 is a pure formula $\Delta = \bigwedge_{i \in I} \delta_i$ such that:*

1. *$\Delta$ is satisfiable, and $\Delta \models \beta(A, B)$;*
2. *$\delta_i$ is of the form $(t < u)$ or $(t = u)$, where $t, u \in \mathcal{T}_{A,B}$, for any $i \in I$;*
3. *for all $t, u \in \mathcal{T}_{A,B}$, there is $i \in I$ such that $\delta_i$ is $(t < u)$ or $(u < t)$ or $(t = u)$.*

**Lemma 5.4.** *Let $\Delta$ be a solution seed for the problem $(A, B)$. $\Delta$ induces a total order on $\mathcal{T}_{A,B}$: for any $e, f \in \mathcal{T}_{A,B}$, $\Delta \models e < f$ or $\Delta \models e = f$ or $\Delta \models f < e$.*

This lemma justifies abbreviating $\Delta \models e < f$ by $e <_\Delta f$; $\Delta \models e \leq f$ by $e \leq_\Delta f$; and, $\Delta \models e = f$ by $e =_\Delta f$.

**Function** $\mathsf{arrcov}_{A,\Delta}(e, f)$
**Data:** a quantifier-free symbolic heap $A$;
solution seed $\Delta$; terms $e$, $f$ in $\mathcal{T}_{A,B}$
**Result:** quantifier-free symbolic heap

*// work with $\mapsto$-abstraction of $A$*

**let** $\left(\Pi : \circledast_{i=1}^{n+k} \mathsf{array}(\hat{a}_i, \hat{b}_i)\right) = \lfloor A \rfloor$;

**if** $f <_\Delta e$ **then**
  *// nothing to cover*
  **return** emp;
**end**

**if** $\exists i \in [1, n+k].\ \hat{a}_i \leq_\Delta e \leq_\Delta \hat{b}_i$ **then**
  *// endpoint $e$ covered by $\mathsf{array}(\hat{a}_i, \hat{b}_i)$*
  **return** $\mathsf{arrcov}_{A,\Delta}(\hat{b}_i + 1, f)$;
**end**

*// left endpoint $e$ not covered*

$E := \left\{ \hat{a}_j \;\middle|\; \begin{array}{l} e <_\Delta \hat{a}_j \leq_\Delta f \\ \text{for } j \in [1, n+k] \end{array} \right\}$;

**if** $E = \emptyset$ **then**
  *// no part of $\mathsf{array}(e, f)$ covered*
  **return** $\mathsf{array}(e, f)$;
**end**

*// middle covered by $\mathsf{array}(\hat{a}_i, \hat{b}_i)$*

$\hat{a}_i := \min_\Delta(E)$;

**return** $(\hat{a}'_i + 1 = \hat{a}_i : \mathsf{array}(e, \hat{a}'_i)) * \mathsf{arrcov}_{A,\Delta}(\hat{b}_i + 1, f)$;

**Function** $\mathsf{ptocov}_{A,\Delta}(e, f)$
**let**
$\left(\Pi : \circledast_{i=1}^{n} \mathsf{array}(a_i, b_i) * \circledast_{i=1}^{k} t_i \mapsto u_i\right) = A$;
**if** $\exists i \in [1, k].\ t_i =_\Delta e$ **then**
  **return** emp;
**end**
**if** $\exists i \in [1, n].\ a_i \leq_\Delta e \leq_\Delta b_i$ **then**
  **return** emp;
**end**
**return** $e \mapsto f$;

– **Arrays of** $A$ / $B$ appear as boxes with indicated bounds.
– **Arrays of** $X$ appear in a hatched pattern.
– **Recursive calls** appear as dashed boxes with parameters.
– **Terms** $a'_i$ are shown as $a_i - 1$ for readability.

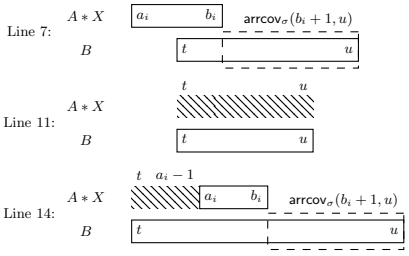



**Fig. 2.** Left: the function $\mathsf{arrcov}_{A,\Delta}(e, f)$. Top right: the function $\mathsf{ptocov}_{A,\Delta}(e, f)$. Bottom right: arrays of $A$, $B$, $X$ relevant to each **return** statement in the arrcov function.

**Theorem 5.5.** *If $\beta(A, B)$ is satisfiable, then there exists a solution seed $\Delta$ for the biabduction problem $(A, B)$.*

*Proof.* (Sketch) Given a stack $s$ such that $s \models \beta(A, B)$, we define the formula

$$\Delta = \bigwedge_{e,f \in \mathcal{T}_{A,B},\ s(e) < s(f)} e < f \ \wedge \ \bigwedge_{e,f \in \mathcal{T}_{A,B},\ s(e) = s(f)} e = f$$

and show that it satisfies Defn. 5.3. □

We now present a way to compute a solution $(X, Y)$ given a solution seed $\Delta$. They key ingredient is the arrcov algorithm (Fig. 2). Intuitively, arrcov takes a solution seed $\Delta$ and the endpoints of an $\mathsf{array}(c_j, d_j)$ in $B$, and constructs arrays for $X$ so that every model of $A * X$ includes a submodel that satisfies $\mathsf{array}(c_j, d_j)$. Arrays in $A$ contribute to the coverage of $\mathsf{array}(c_j, d_j)$ and, in addition, the newly created arrays do not overlap with those of $A$ (or themselves) for consistency.

Note that in arrcov we sometimes need to generate terms denoting the predecessor of the start of an array, even though there is no predecessor function in PbA. We do this by using primed terms $a'_i$, and add constraints that induce this meaning ($a_i + 1 = a'_1$). This is done on demand in order to avoid the risk of trying to decrement a zero-valued term, thus obtaining an inconsistent formula.

**Definition 5.6 (The formulas $X, Y$).** *Let $\Delta$ be a solution seed for an instance $(A, B)$ in the form given in Defn. 5.1. The formulas $X, Y$ are defined as follows:*

$$\Theta_X : F_X =_{def} \bigstar_{j=1}^{m} \mathsf{arrcov}_{A,\Delta}(c_j, d_j) * \bigstar_{j=1}^{\ell} \mathsf{ptocov}_{A,\Delta}(v_j, w_j)$$

$$\Theta_Y : F_Y =_{def} \bigstar_{i=1}^{n} \mathsf{arrcov}_{B,\Delta}(a_i, b_i) * \bigstar_{i=1}^{k} \mathsf{ptocov}_{B,\Delta}(t_i, u_i)$$

$$\hat{\Delta} =_{def} \Delta \wedge \Theta_X \wedge \Theta_Y \qquad X =_{def} \hat{\Delta} : F_X \qquad Y =_{def} \hat{\Delta} : F_Y$$

Every quantifier-free formula $A$ of ASL is *precise* [33] (by structural induction): for any model $s, h$ there exists *at most one* subheap $h'$ of $h$ such that $s, h' \models A$. This motivates the following notation: we will write $[\![A]\!]^{s,h}$ to denote the unique subheap $h' \subseteq h$ such that $s, h' \models A$, when it exists.

**Proposition 5.7.** *Let $(A, B)$ be a biabduction problem of the form shown in Defn. 5.1. Let $\Delta$ be a solution seed and let $e, f$ be terms in $\mathcal{T}_{A,B}$. Then, the call $\mathsf{arrcov}_{A,\Delta}(e, f)$:*

1. *always terminates, issuing up to $n + k$ recursive calls;*
2. *returns a formula $\bigwedge_{i\in I}(a_i = a'_i + 1) \wedge \bigwedge_{i\in J}(t_i = t'_i + 1) : \bigstar_{i=1}^{q} \mathsf{array}(l_i, r_i)$ for some $q \in \mathbb{N}$ and sets $I, J \subseteq \mathbb{N}$, where for all $i \in [1, q]$, $l_i \in \mathcal{T}_{A,B}$;*
3. *for every $i \in [1, q]$, $\hat{\Delta} \models e \leq l_i \leq r_i \leq f$;*
4. *for every $i \in [1, q-1]$, $\hat{\Delta} \models r_i < l_{i+1}$.*

**Lemma 5.8.** *Let $(A, B)$ be a biabduction instance, $\Delta$ a solution seed and $X$ as in Defn. 5.6. Then, $A * X$ is satisfiable.*

**Definition 5.9 ($\mathcal{B}^{\mathsf{arr}}, \mathcal{B}^{\mathsf{pto}}, \mathcal{Y}^{\mathsf{arr}}, \mathcal{Y}^{\mathsf{pto}}$).** *Let $(A, B)$ be a biabduction problem, $\Delta$ a solution seed, $X, Y$ as in Defn. 5.6 and $s, h$ a model such that $s, h \models A * X$. Then we define the following sequences $\mathcal{B}^{\mathsf{arr}}, \mathcal{B}^{\mathsf{pto}}, \mathcal{Y}^{\mathsf{arr}}, \mathcal{Y}^{\mathsf{pto}}$ of subheaps of $h$, such that:*

$$\mathcal{B}^{\mathsf{arr}}_i = [\![\mathsf{array}(c_i, d_i)]\!]^{s,h}\ i \in [1, m] \qquad \mathcal{Y}^{\mathsf{arr}}_i = [\![\mathsf{arrcov}_{B,\Delta}(a_i, b_i)]\!]^{s,h}\ i \in [1, n]$$
$$\mathcal{B}^{\mathsf{pto}}_i = [\![v_i \mapsto w_i]\!]^{s,h}\ i \in [1, \ell] \qquad \mathcal{Y}^{\mathsf{pto}}_i = [\![\mathsf{ptocov}_{B,\Delta}(t_i, u_i)]\!]^{s,h}\ i \in [1, k]$$

**Lemma 5.10.** *All heaps in $\mathcal{B}^{\mathsf{arr}}, \mathcal{B}^{\mathsf{pto}}, \mathcal{Y}^{\mathsf{arr}}, \mathcal{Y}^{\mathsf{pto}}$ are well-defined. Also,*

1. *For any $\mathcal{S}$ of $\mathcal{B}^{\mathsf{arr}}$,$\mathcal{B}^{\mathsf{pto}}$,$\mathcal{Y}^{\mathsf{arr}}$,$\mathcal{Y}^{\mathsf{pto}}$, and any distinct $i, j \in [1, |\mathcal{S}|]$, $\mathcal{S}_i \# \mathcal{S}_j$.*
2. *For any two distinct $\mathcal{S}, \mathcal{T}$ of $\mathcal{B}^{\mathsf{arr}}$, $\mathcal{B}^{\mathsf{pto}}$, $\mathcal{Y}^{\mathsf{arr}}$, $\mathcal{Y}^{\mathsf{pto}}$, and any $i, j$, $\mathcal{S}_i \# \mathcal{T}_j$.*
3. *$\mathrm{dom}\,(h) \subseteq \bigcup_{i=1}^{m} \mathcal{B}^{\mathsf{arr}}_i \cup \bigcup_{i=1}^{\ell} \mathcal{B}^{\mathsf{pto}}_i \cup \bigcup_{i=1}^{n} \mathcal{Y}^{\mathsf{arr}}_i \cup \bigcup_{i=1}^{k} \mathcal{Y}^{\mathsf{pto}}_i$.*

**Theorem 5.11.** *Given a solution seed $\Delta$ for the biabduction problem $(A, B)$, the formulas $X$ and $Y$, as computed by Defn. 5.6, form a solution for that instance.*

*Proof.* That $(X, Y)$ is a solution means that $A * X$ is satisfiable and that $A * X \models B * Y$. The first requirement is fulfilled by Lemma 5.8. Here, we show the second.

Let $s, h$ be a model of $A * X$. We need to show that $s, h \models B * Y$. Using Defn. 5.6, we have $A * X = \Pi \wedge \hat{\Delta} : F_{A*X}$ and $B * Y = \Pi' \wedge \hat{\Delta} : F_{B*Y}$. It is easy to see that $s \models \Pi' \wedge \hat{\Delta}$: by assumption, $s \models \hat{\Delta}$, and as $\hat{\Delta} \models \Delta$ (Defn. 5.6) and $\Delta \models \gamma(B)$ (Defn. 5.3), it follows that $s \models \Pi'$ as well (Defn. 4.4).

It remains to show that $s, h \models F_{B*Y}$. We will do this by (a) defining a subheap $h' \subseteq h$ for each atomic formula $\sigma$ in $F_{B*Y}$, such that $s, h' \models \sigma$. Having done this we will need (b) to show that all such subheaps are disjoint, and that (c) their disjoint union equals $h$.

The sequences $\mathcal{B}^{\mathsf{arr}}, \mathcal{B}^{\mathsf{pto}}, \mathcal{Y}^{\mathsf{arr}}, \mathcal{Y}^{\mathsf{pto}}$ from Defn. 5.9, by construction, fulfil requirement (a) above, given they are well-defined as guaranteed by Lemma 5.10 (main statement). Requirement (b) is covered by items 1 and 2 of Lemma 5.10. Finally, requirement (c) is covered by item 3 of Lemma 5.10. □

The solutions obtained via Defn. 5.6 are constructed from terms in $\mathcal{T}_{A,B}$, so $X, Y$ are as 'symbolic' as $A, B$ are. However, our solutions are potentially stronger than required; our algorithm here always imposes a total order over all array endpoints in the antiframe $X$, even if only a *part* of this information is actually required in order to compute the spatial formulas in $X$ and $Y$. We believe that our algorithm can be refined so as to avoid "over-committing".

Our method is, also, complete in the following sense. Suppose $(X, Y)$ is a solution that does not impose a total order over $\mathcal{T}_{A,B}$. Then, there exists a solution $(X', Y')$ computable by our method, such that $X' \models X$ and $Y' \models Y$.

### 5.2 Complexity of quantifier-free biabduction in ASL

**Lemma 5.12.** *Let $(A, B)$ be a biabduction instance and $\Delta$ a formula satisfying Conditions 2 and 3 of Defn. 5.3. Let $\Gamma = \bigwedge \bigvee \pi$ be a formula where $\pi$ is of the form $t < u$ or $t = u$ and $t, u \in \mathcal{T}_{A,B}$. Then, checking $\Delta \models \Gamma$ is in* PTIME.

**Proposition 5.13.** *Deciding if there is a solution for a biabduction problem $(A, B)$, and constructing it if it exists, can be done in* NP.

*Proof.* (Sketch) We guess a total order over $\mathcal{T}_{A,B}$ and a polynomially-sized assignment of values $s$ ([35, Theorem 6]) to all terms in $\mathcal{T}_{A,B}$. We convert this order to a formula $\Delta$ and check if $s \models \Delta$ (thus showing the satisfiability of $\Delta$) and whether $\Delta \models \beta(A, B)$. If all these conditions hold, we use Defn. 5.6 and obtain formulas $X, Y$. By Prop. 5.7 and Lemma 5.12 this process runs in PTIME. □

We establish NP-hardness of quantifier-free biabduction by reduction from the 3-partition problem, similarly to satisfiability in Section 4.

**Definition 5.14.** *Let $(B, \mathcal{S})$ be an instance of the 3-partition problem. We define corresponding symbolic heaps $\widetilde{A}_{B,\mathcal{S}}$ and $\widetilde{B}_{B,\mathcal{S}}$. First, we define a satisfiable $\widetilde{A}_{B,\mathcal{S}}$ as: $\bigwedge_{i=1}^{m}(d_{i+1} = d_i + B + 1) : \mathop{\ast}_{i=1}^{m+1} \mathsf{array}(d_i, 0, 0)$. The formula $\widetilde{B}_{B,\mathcal{S}}$, a relaxed but satisfiable version of $A_{B,\mathcal{S}}$ from Defn 4.7, is given by:*

$$\bigwedge_{i=1}^{m} d_{i+1} > d_i \wedge \bigwedge_{j=1}^{3m}(d_1 \leq a_j \wedge a_j + k_j < d_{m+1}) : \mathop{\ast}_{i=1}^{m+1} \mathsf{array}(d_i, 0, 0) * \mathop{\ast}_{j=1}^{3m} \mathsf{array}(a_j, 1, k_j)$$

**Lemma 5.15.** *Let $A_{B,\mathcal{S}}$ be the symbolic heap given by Definition 4.7. Then we have the Presburger equivalence $\beta(\widetilde{A}_{B,\mathcal{S}}, \widetilde{B}_{B,\mathcal{S}}) \equiv \gamma(A_{B,\mathcal{S}})$.*

*Proof.* (Sketch) Follows from Defns. 5.14, 5.1 and 4.4. □

**Theorem 5.16.** *The biabduction problem for* ASL *is* NP*-hard, even for $(A, B)$ such that $A, B$ are satisfiable, quantifier-free, and in two-variable form.*

*Proof.* (Sketch) By reduction from the 3-partition problem (see Section 4). By Lemmas 4.5, 4.8 and 5.15, there is a complete 3-partition on $\mathcal{S}$ w.r.t. bound $B$ iff $\beta(\widetilde{A}_{B,\mathcal{S}}, \widetilde{B}_{B,\mathcal{S}})$ is satisfiable. Using (Prop. 5.2 / Thm. 5.5 / Thm. 5.11), this is equivalent to the existence of a biabduction solution for $(\widetilde{A}_{B,\mathcal{S}}, \widetilde{B}_{B,\mathcal{S}})$. □

### 5.3 Biabduction for ASL with quantifiers

Here we show two complementary results about biabduction when $B$ contains existential quantifiers. First, if the quantifiers are appropriately restricted, then the biabduction problem is equivalent to the quantifier-free case (thus in NP). If quantifiers are *not* restricted, then the problem becomes $\Pi_2^P$-hard [36].

**Proposition 5.17.** *Let $A$ be quantifier-free, and let $B$ be such that no variable appearing in the RHS of a $\mapsto$ formula is existentially bound. Then, a biabduction instance $(A, B)$ has a solution if and only if $(A, \mathsf{qf}(B))$ has a solution.*

The construction of a suitable heap $h$ in the proof of the nontrivial ($\Rightarrow$) direction of Prop. 5.17 explains the reasons for our restrictions on quantifiers: the contents of the arrays in $h$ must be chosen different to the data values occurring in the $\mapsto$-formulas in $B$. If any such values are quantified, this may be impossible. Indeed, $X = Y = \mathsf{emp}$ is a trivial biabduction solution for $\mathsf{array}(x, x) * X \models (\exists y.\ x \mapsto y) * Y$, but no solution exists if we remove the quantifier.

In order to obtain the $\Pi_2^P$ lower bound for biabduction with unrestricted quantifiers, we reduce from the following *colourability* problem, from [1].

**2-round 3-colourability problem.** *Let $G = (V, E)$ be an undirected graph with $n$ vertices $v_1, \ldots, v_k, v_{k+1}, \ldots v_n$, and let $v_1, v_2, \ldots, v_k$ be its* leaves*. The problem is to decide if every 3-colouring of the leaves can be extended to a 3-colouring of the graph, such that no two adjacent vertices share the same colour.*

Let $c_{i,1}$ denote the colour, 1,2, or 3, the vertex $v_i$ is marked by. We mark also each edge $(v_i, v_j)$ by $\widetilde{c_{ij}}$, the colour "complementary" to $c_{i,1}$ and $c_{j,1}$.

As for the leaves $v_i$, we introduce $k$ distinct locations $d_1, \ldots, d_k$ so that the value $c_i$ stored in $d_i$ can be used subsequently to identify the colour $c_{i,1}$ marking $v_i$, e.g., with the help of $(c_{i,1} - 1 \equiv c_i \pmod 3)$ .

We encode the fact that $c_{i,1}$, $c_{j,1}$, and $\widetilde{c_{ij}}$ are distinct by taking $c_{i,1}$, $c_{j,1}$, and $\widetilde{c_{ij}}$ as the addresses, adjusted with a base-offset $e_{ij}$, for three consecutive cells within a memory chunk of length 3 given by $\mathsf{array}(e_{ij}, 1, 3)$, which forces these colours to form a *permutation* of $(1, 2, 3)$.

**Definition 5.18.** *An arbitrary* 3*-colouring of the leaves is encoded with a satisfiable* $A_G$ *taken as*

$$A_G =_{def} (b = 3) \colon \mathop{\ast}_{i=1}^{k} \mathsf{array}(d_i, 1, 1) * \mathop{\ast}_{(v_i,v_j)\in E} \mathsf{array}(e_{ij}, 1, 3) \ .$$

*For a fixed* $b$*, a perfect* $b$*-colouring of the whole* $G$ *is encoded with* $B_G$ *taken as*

$$\exists \mathbf{z}. \ \big( \bigwedge_{i=1}^{n} (1 \leq c_{i,1} \leq b) \wedge \bigwedge_{(v_i,v_j)\in E} (1 \leq \widetilde{c_{ij}} \leq b) \wedge \bigwedge_{i=1}^{k} (c_{i,1} - 1 \equiv c_i \ (\mathrm{mod} 3)) \colon$$
$$\mathop{\ast}_{i=1}^{k} d_i \mapsto c_i \ * \mathop{\ast}_{(v_i,v_j)\in E} \mathsf{array}(e_{ij}, c_{i,1}, c_{i,1}) * \mathsf{array}(e_{ij}, c_{j,1}, c_{j,1}) * \mathsf{array}(e_{ij}, \widetilde{c_{ij}}, \widetilde{c_{ij}}) \big).$$

*where the existentially quantified variables* $\mathbf{z}$ *are all variables occurring in* $B_G$ *that are not mentioned explicitly in* $A_G$*.*

$B$ is *satisfiable*, e.g., for a large $b$, each vertex $v_i$ can be marked by its own colour.

**Lemma 5.19.** *Let* $G$ *be a* 2*-round* 3*-colouring instance. The biabduction problem* $(A_G, B_G)$ *has a solution iff there is a winning strategy for the perfect* 3*-colouring* $G$*, where* $A_G$ *and* $B_G$ *are the symbolic heaps given by Defn. 5.18.*

**Theorem 5.20.** *The biabduction problem* $(A, B)$ *for* $\mathsf{ASL}$ *is* $\Pi_2^P$*-hard, even if* $A$ *is quantifier-free and both* $A$ *and* $B$ *are satisfiable.*

*Proof.* Follows from Lemma 5.19. □

## 6 Entailment

We now focus on *entailment* for $\mathsf{ASL}$. We establish an upper bound of $\Pi_1^{\mathrm{EXP}}$ in the *weak* $\mathsf{EXP}$ *hierarchy* [26] via an encoding into $\Pi_2^0$ $\mathsf{PbA}$, and a lower bound of $\Pi_2^P$ [36]. Moreover, quantifier-free entailment is $\mathsf{coNP}$-complete.

**Entailment for** $\mathsf{ASL}$**.** *Given symbolic heaps* $A, B$*, decide if* $A \models B$*. A may be considered quantifier-free; similar to Prop. 5.17, the existential quantifiers in* $B$ *may not mention variables appearing in the RHS of a* $\mapsto$*-formula.*

The intuition underlying our encoding of entailment: There exists a countermodel for $A \models B$ iff there exists a stack $s$ that induces a model for $A$ (captured by $\gamma(A)$ from Section 4) and, for every instantiation of the existentially quantified variables in $B$ (say $\mathbf{z}$), one of the following holds under $s$:

1. the quantifier-free body $\mathsf{qf}(B)$ of $B$ becomes unsatisfiable; or
2. some heap location is covered by an array or pointer in $A$, but not by any array or pointer in $B$, or vice versa; or

3. the LHS of some pointer in B is covered by an array in $A$ (thus we can choose the contents of the array different to the contents of the pointer); or
4. a pointer in $B$ is covered by a pointer in $A$, but their data contents disagree.

Similar to Prop. 5.17, this intuition explains our restriction on quantification in the entailment problem: if we allow quantifiers over the RHS of $\mapsto$ formulas, then item 3 above might or might not be sufficient to construct a countermodel. For example, there is a countermodel for $\mathsf{array}(x, x) \models \exists y.\ y \leq 3 : x \mapsto y$, and for $\mathsf{array}(x, x) \models x \mapsto y$, but not for $\mathsf{array}(x, x) \models \exists y.\ x \mapsto y$.

**Definition 6.1.** *Let $A$ and $B$ be two $\mapsto$-free symbolic heaps such that*

$$A = \mathsf{array}(a_1, b_1) * \ldots * \mathsf{array}(a_n, b_n)$$
$$B = \mathsf{array}(c_1, d_1) * \ldots * \mathsf{array}(c_m, d_m)$$

*Then we define the formula $\phi(A, B)$ of* PbA *to be*

$$\phi(A, B) =_{def} \exists x.\ \textstyle\bigvee_{i=1}^{n} a_i \leq x \leq b_i \wedge \bigwedge_{j=1}^{m} (x < c_j) \vee (x > d_j)\ ,$$

*where $x$ is fresh. We lift $\phi$ to arbitrary symbolic heaps by ignoring quantifiers and abstracting pointers to arrays using $\lfloor - \rfloor$, i.e., $\phi(A, B) = \phi(\lfloor \mathsf{qf}(A) \rfloor, \lfloor \mathsf{qf}(B) \rfloor)$.*

**Lemma 6.2.** *We can rewrite $\phi(A, B)$ as a quantifier-free formula in polytime.*

**Definition 6.3.** *Let $A$ and $B$ be symbolic heaps with $A$ quantifier-free:*

$$A = \Pi : \mathop{\Large *}_{i=1}^{n} \mathsf{array}(a_i, b_i) * \mathop{\Large *}_{i=1}^{k} t_i \mapsto u_i$$
$$B = \exists \mathbf{z}.\ \Pi' : \mathop{\Large *}_{j=1}^{m} \mathsf{array}(c_j, d_j) * \mathop{\Large *}_{j=1}^{\ell} v_j \mapsto w_j$$

*where the existentially quantified variables $\mathbf{z}$ are disjoint from all variables in $A$. We define formulas $\psi_1(A, B)$, $\psi_2(A, B)$ and $\chi(A, B)$ of* PbA *as follows:*

$$\psi_1(A, B) = \textstyle\bigvee_{i=1}^{n} \bigvee_{j=1}^{\ell} a_i \leq v_j \leq b_i\ ,$$
$$\psi_2(A, B) = \textstyle\bigvee_{i=1}^{k} \bigvee_{j=1}^{\ell} (t_i = v_j) \wedge (u_i \neq w_j)\ ,\ \text{and}$$
$$\chi(A, B) = \gamma(A) \wedge \forall \mathbf{z}.\big(\neg\gamma(\mathsf{qf}(B)) \vee \phi(A, B) \vee \phi(B, A) \vee \psi_1(A, B) \vee \psi_2(A, B)\big)$$

*where $\gamma(-)$ is given by Defn. 4.4, and $\phi(-, -)$ by Defn. 6.1.*

**Lemma 6.4.** *For any instance $(A, B)$ of the* ASL *entailment problem above, and for any stack $s$, we have $s \models \chi(A, B)$ iff $\exists h.\ s, h \models A$ and $s, h \not\models B$.*

**Theorem 6.5.** *Entailment is in $\Pi_1^{\mathrm{EXP}}$. If the no. of variables in $A, B$ is fixed, the problem is in $\Pi_2^P$, and if $B$ is quantifier-free then the problem is in* coNP.

*Proof.* Follows from Lemmas 6.2 and 6.4, plus relevant complexity results for Presburger arithmetic [23,25,36]. □

In order to obtain the $\Pi_2^P$ lower bound for entailment, we exhibit a reduction from the same colourability problem as in Section 5.3. See [11] for details.

**Theorem 6.6.** *The entailment problem $A \models B$ is $\Pi_2^P$-hard, even when $A$ is quantifier-free, and $A, B$ are satisfiable symbolic heaps in two-variable form. Moreover, the entailment problem is* coNP*-hard even for quantifier-free symbolic heaps in two-variable form.*

*Proof.* For the general case, we reduce from the 2-round 3-colourability problem, which is $\Pi_2^P$-hard [1]. For the quantifier-free case, the upper bound is immediate by Thm. 6.5. For the lower bound, consider the entailment $A_{B,\mathcal{S}} \models x < x : \mathsf{emp}$ where $(B, \mathcal{S})$ is a 3-partition instance (see Section 4) and $A_{B,\mathcal{S}}$ is the symbolic heap in two-variable form given by Defn. 4.7. Using Lemma 4.8, this entailment is valid iff there is *no* complete 3-partition on $\mathcal{S}$ w.r.t. $B$, a coNP-hard problem. □

In the general case, there is a gap between our upper and lower bounds for entailment: $\Pi_1^{\mathrm{EXP}} = \mathsf{coNEXP}$ versus $\Pi_2^P = \mathsf{coNP}^{\mathsf{NP}}$, respectively. It is plausible that the lower bound is at least EXP: however, an encoding of, e.g., $\Pi_0^2$ PbA in ASL is not straightforward, because our pure formulas are conjunctions rather than arbitrary Boolean combinations of atomic Presburger formulas.

Nevertheless, we note the essential difference between the biabduction and entailment problems for ASL: by Theorem 6.6, entailment is still $\Pi_2^P$-hard whereas, by Props. 5.13 and 5.17, biabduction is in NP.

## 7 Related work

Here we briefly survey the literature most closely related to the present paper. A fuller discussion appears in [11].

First, symbolic-heap separation logic over linked lists [4], underpinning the Infer tool [13], has been extensively studied; its satisfiability and entailment problems have been shown to be in PTIME [16], and its abduction problem (where only an "antiframe" $X$ is computed) is known NP-complete [22]. The biabduction problem is studied in [14]. However, this fragment and our ASL are largely disjoint: our arrays cannot be defined in terms of list segments or vice versa, while ASL also employs linear arithmetic rather than simple (dis)equalities. This is also reflected in the differences in their respective complexity bounds.

Moreover, even when arbitrary inductive definitions over symbolic heaps are permitted [9], an area that has received significant recent interest (see e.g. [3,10,12,27,38]) our ASL cannot be encoded in the absence of arithmetic. Very recently, in [24], decidability of satisfiability and entailment was obtained for a fragment of symbolic-heap separation logic with ("linearly compositional") inductive predicates *and* arithmetic. However, ASL cannot be encoded in this fragment, because pointers and data variables belong to disjoint sorts, effectively disallowing pointer arithmetic. A semidecision procedure for satisfiability in symbolic-heap separation logic with inductive definitions and Presburger arithmetic appears in [30]. ASL can be encoded in their logic, but, as far as we can tell, not into the subfragment for which they show satisfiability decidable.

The *iterated separating conjunction* (ISC) [34], a binding operator for expressing various unbounded data structures, was recognised early on as a way

of reasoning about arrays. E.g., [31] uses the ISC to reasoning about memory permissions, with the aim of enabling symbolic execution of concurrent array-manipulating program. However, although our array predicate can be expressed using the ISC, we do not know of any existing decision procedures for biabduction, entailment or even satisfiability in such a logic, which may be of high complexity or become undecidable. We note for example that the analysis in [31] requires programs to be fully annotated.

Finally, a significant amount of research effort has previously focused on the verification of array-manipulating programs either via invariant inference and theorem proving, or via abstract interpretation (for instance [29,20,19,7,2,37]). These approaches differ from ours technically, but also in intention. First, the emphasis in these investigations is on data constraints and, thus, tends towards proving general safety properties of programs. Here, we intentionally restrict the language so that we can obtain sound and complete algorithms which can be used for establishing memory safety of programs but not for proving arbitrary safety properties. Second, such approaches are typically whole-program analyses that cannot be used bottom-up (with the possible exception of the non-SL-based [6,18]). In contrast, our focus is on biabduction, one of the key ingredients that makes such a compositional approach possible.

## 8 Conclusions and future work

In this paper, we investigate ASL, a separation logic aimed at compositional memory safety proofs for array-manipulating programs. We give a sound and complete NP algorithm for the crucial *biabduction* problem in this logic, and we show that the problem is NP-hard in the quantifier-free case. In addition, we show that the satisfiability problem for ASL is NP-complete, and entailment is decidable, being coNP-complete for quantifier-free formulas, and at least $\Pi_2^P$-hard (perhaps much harder) in general.

An obvious direction for future work is to build an abductive program analysis à la Infer [13] for array programs, using ASL as the assertion language. An outstanding issue is finding biabduction solutions that are as logically weak as possible; our algorithm currently commits to a total ordering of all arrays even if a partial ordering would be sufficient. We believe that, in practice, this could be resolved by refining the notion of a solution seed so that it carries *just* enough information for computing the spatial formulas.A more conceptually interesting problem is how we might assess the quality of logically incomparable solutions.

In addition, a program analysis for ASL will rely not just on biabduction but also on suitable *abstraction* heuristics for discovering loop invariants; this seems an interesting and non-trivial problem for the near future.

Another possible direction for future work is on combining ASL with other fragments of separation logic, such as the linked list fragment, for increased expressivity. We are uncertain whether our techniques would extend naturally to such logics, but we consider this a very interesting area for future study.

## References


1. Ajtai, M., Fagin, R., Stockmeyer, L.J.: The closure of monadic NP. J. Comput. Syst. Sci. 60(3), 660–716 (2000)
2. Alberti, F., Ghilardi, S., Sharygina, N.: Decision procedures for flat array properties. In: Proc. TACAS-20. pp. 15–30. Springer (2014)
3. Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M., Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates. In: Proc. FoSSaCS-17. pp. 411–425. Springer (2014)
4. Berdine, J., Calcagno, C., O'Hearn, P.: A decidable fragment of separation logic. In: Proc. FSTTCS-24. LNCS, vol. 3328, pp. 97–109. Springer (2004)
5. Berdine, J., Cook, B., Ishtiaq, S.: SLAyer: memory safety for systems-level code. In: Proc. CAV-23. pp. 178–183. Springer (2011)
6. Bouajjani, A., Drăgoi, C., Enea, C., Sighireanu, M.: A logic-based framework for reasoning about composite data structures. In: Proc. CONCUR-20. pp. 178–195. Springer (2009)
7. Bouajjani, A., Drăgoi, C., Enea, C., Sighireanu, M.: Accurate invariant checking for programs manipulating lists and arrays with infinite data. In: Proc. ATVA-10. LNCS, vol. 7561, pp. 167–182. Springer-Verlag (2012)
8. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Proc. VMCAI-7. pp. 427–442. Springer (2006)
9. Brotherston, J.: Formalised inductive reasoning in the logic of bunched implications. In: Proc. SAS-14. LNCS, vol. 4634, pp. 87–103. Springer-Verlag (2007)
10. Brotherston, J., Fuhs, C., Gorogiannis, N., Navarro Pérez, J.: A decision procedure for satisfiability in separation logic with inductive predicates. In: Proc. CSL-LICS. pp. 25:1–25:10. ACM (2014)
11. Brotherston, J., Gorogiannis, N., Kanovich, M.: Biabduction (and related problems) in array separation logic. CoRR abs/1607.01993 (2016), `http://arxiv.org/abs/1607.01993`
12. Brotherston, J., Gorogiannis, N., Kanovich, M., Rowe, R.: Model checking for symbolic-heap separation logic with inductive predicates. In: Proc. POPL-43. pp. 84–96. ACM (2016)
13. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: Proc. NFM-7. LNCS, vol. 9058, pp. 3–11. Springer (2015)
14. Calcagno, C., Distefano, D., O'Hearn, P., Yang, H.: Compositional shape analysis by means of bi-abduction. J. ACM 58(6) (December 2011)
15. Chin, W.N., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comp. Prog. 77(9), 1006–1036 (2012)
16. Cook, B., Haase, C., Ouaknine, J., Parkinson, M., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: Proc. CONCUR-22. LNCS, vol. 6901, pp. 235–249. Springer (2011)
17. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, 3rd edn. (2009)
18. Cousot, P., Cousot, R., Fähndrich, M., Logozzo, F.: Automatic inference of necessary preconditions. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Proc. VMCAI-14. pp. 128–148. Springer (2013)
19. Cousot, P., Cousot, R., Logozzo, F.: A parametric segmentation functor for fully automatic and scalable array content analysis. In: Proc. POPL-38. pp. 105–118. ACM (2011)

20. Dillig, I., Dillig, T., Aiken, A.: Fluid updates: Beyond strong vs. weak updates. In: Proc. ESOP-19. pp. 246–266. Springer (2010)
21. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
22. Gorogiannis, N., Kanovich, M., O'Hearn, P.W.: The complexity of abduction for separated heap abstractions. In: Proc. SAS-18. LNCS, vol. 6887, pp. 25–42. Springer (2011)
23. Grädel, E.: Subclasses of Presburger arithmetic and the polynomial-time hierarchy. Theoretical Computer Science 56, 289–301 (1988)
24. Gu, X., Chen, T., Wu, Z.: A complete decision procedure for linearly compositional separation logic with data constraints. In: Proc. IJCAR. LNAI, vol. 9706, pp. 532–549. Springer (2016)
25. Haase, C.: Subclasses of Presburger arithmetic and the weak EXP hierarchy. In: Proceedings of CSL-LICS. pp. 47:1–47:10. ACM (2014)
26. Hartmanis, J., Immerman, N., Sewelson, V.: Sparse sets in NP-P: EXPTIME versus NEXPTIME. Inform. Control 65(2), 158 – 181 (1985)
27. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: Proc. CADE-24. LNAI, vol. 7898, pp. 21–38. Springer (2013)
28. Jacobs, B., Smans, J., Philippaerts, P., Vogels, F., Penninckx, W., Piessens, F.: VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In: Proc. NFM-3. LNCS, vol. 6617, pp. 41–55. Springer (2011)
29. Kovács, L., Voronkov, A.: Finding loop invariants for programs over arrays using a theorem prover. In: Proc. FASE-12. pp. 470–485. Springer (2009)
30. Le, Q.L., Sun, J., Chin, W.N.: Satisfiability modulo heap-based programs. In: Proc. CAV-28 (2016)
31. Müller, P., Schwerhoff, M., Summers., A.J.: Automatic verification of iterated separating conjunctions using symbolic execution. In: Proc. CAV-28 (to appear, 2016)
32. O'Hearn, P.W.: Resources, concurrency and local reasoning. Theoretical Computer Science 375(1–3), 271–307 (2007)
33. O'Hearn, P.W., Yang, H., Reynolds, J.C.: Separation and information hiding. In: Proc. POPL-31. pp. 268–280. ACM (2004)
34. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Proc. LICS-17. pp. 55–74. IEEE (2002)
35. Scarpellini, B.: Complexity of subcases of Presburger arithmetic. Trans. American Mathematical Society 284(1), 203–218 (1984)
36. Stockmeyer, L.J.: The polynomial-time hierarchy. Theoretical Computer Science 3, 1–22 (1977)
37. Ströder, T., Giesl, J., Brockschmidt, M., Frohn, F., Fuhs, C., Hensel, J., Schneider-Kamp, P., Aschermann, C.: Automatically proving termination and memory safety for programs with pointer arithmetic. Journal of Automated Reasoning To appear
38. Tatsuta, M., Kimura, D.: Separation logic with monadic inductive definitions and implicit existentials. In: Proc. APLAS-13. LNCS, vol. 9458, pp. 69–89. Springer (2015)
39. Yang, H., O'Hearn, P.: A semantic basis for local reasoning. In: Proc. FOSSACS-5. pp. 402–416. Springer (2002)