Bi-Abductive Inference for Shape and Ordering Properties

Christopher Curry
School of Computing
and Digital Technologies
Teesside University
Middlesbrough
Tees Valley TS1 3BX, UK
Email: c.curry@tees.ac.uk

Quang Loc Le
School of Computing
and Digital Technologies
Teesside University
Middlesbrough
Tees Valley TS1 3BX, UK
Email: q.le@tees.ac.uk

Shengchao Qin
School of Computing
and Digital Technologies
Teesside University
Middlesbrough
Tees Valley TS1 3BX, UK
Email: s.qin@tees.ac.uk

***Abstract*— In separation logic, bi-abduction - a combination of abductive inference and frame inference - is the key enabler for compositional reasoning, helping to scale up verification significantly. Indeed, the success of bi-abduction led to the development of Infer, the tool used daily to verify Facebook's codebase of millions of lines of code. However, this success currently stays largely within the shape domain. To extend this impact towards the combination of shape and arithmetic domains, in this work, we present a novel one-stage bi-abductive procedure for a combination of data structures and ordering values. The procedure is designed in the spirit of the Unfold-and-Match paradigm where the inference is utilized to derive any mismatched portion. We demonstrate our proposal through several interesting examples to show that it is promising for an automated verification of heap-manipulating programs.**

***Index Terms*—Bi-abduction, Separation Logic, Specification Inference.**

I. Introduction

Heap-manipulation is a powerful building block used in many real-world applications and is frequently utilised in low-level system software such as device drivers. While useful, heap manipulation can also be a dangerous technique, one in which relatively simple faults can cause significant numbers of issues, ranging from software crashes [1] to security vulnerabilities [2]. As a result, ensuring the correctness and safety of such programs is of great importance, yet such analyses have been shown to be highly difficult tasks.

In recent years, shape analysis has emerged as a strong candidate for accomplishing such verification tasks, and with the integration of separation logic [3], [4] into shape analysis techniques, an increasing number of advanced and usable tools are being developed in the area of automatic reasoning about dynamically allocated heap programs.

One of the most promising techniques currently utilised in this area is bi-abductive inference [5], [6], [7]. First introduced in [5], bi-abductive inference, or bi-abduction for short, is the combination of the abductive inference and frame inference techniques into a singular analysis. Given two formulas in separation logic, A and C, bi-abduction aims to identify some non-trivial terms $?M$ and $?F$ such that the entailment

$$A * [?M] \vdash C * [?F]$$

(where $*$ is the separating conjunction) is satisfied.

Bi-abduction has seen a large growth in interest in recent years due to a number of useful properties of the technique. First, bi-abductive techniques generally have very low requirements placed upon the end-user, with many requiring only the code of the program to be analysed. Second, bi-abduction is a compositional analysis: an analysis in which the final output is the combination of the results of smaller analyses over components of the program, typically procedures. Compositional analyses provide a number of significant benefits, including high scalability, parallelisation, the ability to undertake incremental analyses and potential support for graceful failure, all highly desirable properties for an analysis technique. More recent work has also included advances that support the use of the technique over near-arbitrary data structures [7], further improving the usability of the technique. However, while bi-abduction has proven to be a highly capable approach to program analysis, the majority of existing implementations are limited to the shape domain only. This restriction not only limits the range of inputs that can be supported by the technique, but also carries the potential risk of imprecision when used to analyse data structures that have pure constraints as a component of their design, such as binary search trees or sorted linked lists. Currently, efforts are being made towards extending bi-abduction to the combined domain, allowing bi-abductive techniques to operate over general pure constraints in addition to those shape properties, with common examples of investigated pure constraints including size, ordering and content (bag) properties.

As an example, the technique outlined by Trinh et al [8] utilises an enrichment-based approach to the identification of pure constraints, but is fully reliant upon some previous shape analysis in order to function. This multi-phase approach is common in the literature of combined domain verification techniques [9], and while it has been shown to be effective, the approach is apparently not very efficient. Additionally, by separating the shape and pure domain, the accuracy of the final inferred specification would be heavily reliant upon the accuracy and expressiveness of the shape information determined in the initial analysis, potentially negatively affecting

$x, y, ... \in Var$	variables
$E ::= \texttt{null} \mid x$	expressions
$V ::=$ i, j, ... $\in Values$	values
$P ::= E{=}E \mid E{<}E$	simple pure formulae
$\Pi ::= \texttt{true} \mid P \mid \neg\Pi \mid$ $\exists v.\Pi \mid \Pi \wedge \Pi$	pure formulae
$f, f_i, ... \in Fields$	fields
$\rho ::= f_1 : E_1, ..., f_k : E_k$	record expressions
$\Sigma ::= \texttt{emp} \mid E{\mapsto}[\rho] \mid \Sigma * \Sigma \mid$ $ls(E, E) \mid sls(E, x, y, E)$	
$\Delta ::= \Pi \wedge \Sigma$	qf symbolic heaps
$H ::= \exists \overrightarrow{X}.\Delta$	symbolic heaps

(a) Grammar Definitions

$$ls(E_1, E_2) \stackrel{def}{=} E_1 = E_2 \wedge \texttt{emp} \vee \exists E'.E_1 \neq E_2 \wedge E_1{\mapsto}[E'] * ls(E', E_2)$$

$$sls(E_1, V_1, V_2, E_2) \stackrel{def}{=} E_1{=}E_2 \wedge V_1{=}V_2 \wedge \texttt{emp} \vee \exists E', V.V_1{\leq}V \wedge E_1{\neq}E_2 \wedge E_1{\mapsto}[E', V_1] * sls(E', V, V_2, E_2)$$

(b) Predicate Definitions

Figure 1: Language Fragment

the quality of the overall result.

In this work, we outline a novel one-phase bi-abduction procedure for the combination of shape and ordering properties. We present a (sub)set of inference rules developed using the rules of Smallfoot [10] as a starting point and a novel algorithm to search for a sequence of rule applications, and demonstrate this initial system over a small number of examples.

II. Preliminaries

The language used in our work is shown in Figure 1a. Program variables are defined as $italic$ characters, and refer to variables originating from within the program itself. Logical variables are indicated with upper-case letters, such as X, and refer to variables that appear in the analysis only.

We often omit record fields from the notation where unambiguous: $x{\mapsto}[n{:}y, v{:}z]$ may be shortened to $x{\mapsto}[y, z]$, for example. We may additionally omit the square brackets around single-field records, as in $x{\mapsto}y$.

Our language includes inductive definitions describing fundamental list structures: ls, representing a simple singly-linked list, and sls, representing a *sorted* singly-linked list. The full definitions may be found in Figure 1b.

A sorted singly-linked list $sls(E_1, V_1, V_2, E_2)$ is a sequence of singly-linked nodes, beginning with node E_1 and ending at some node E_2 (non-inclusive), with all values stored in those nodes obeying an ascending order. In order to simplify the checking of these structures, the minimum and maximum values of the list are also tracked inside the predicate, with the minimum value being V_1 and the maximum V_2. Note that V_2 *is not* the value of node E_2; rather, V_2 would refer to the value of the final node in the list, which points to E_2.

The semantics of this fragment is quite standard, following from the semantics of separation logic with general inductive definitions and arithmetic presented in [11].

The semantics is given by a relation $s,h \models H$ that forces the stack s and heap h to satisfy the constraint H where $h \in$ *Heaps*, $s \in$ *Stacks*, and H is a formula. Stack and heap abstractions are defined:

$$\begin{array}{rcl} \textit{Heaps} & \stackrel{\text{def}}{=} & \textit{Values} \rightharpoonup_{fin} (\textit{Fields} \times \textit{Values})^N \\ \textit{Stacks} & \stackrel{\text{def}}{=} & \textit{Var} \rightarrow \textit{Values} \end{array}$$

where N is the maximum number of fields.

Note that we preserve $s(\texttt{null})$ as a special value such that it is not in any domain of heaps.

$s, h \models \texttt{emp}$	*iff*	$dom(h){=}\{\}$
$s, h \models E{\mapsto}[f_i{:}v_i]$	*iff*	$dom(h){=}\{s(E)\}$ $h(s(E)){=}((f_1, s(E_1)), .., (f_N, s(E_N)))$
$s, h \models \Sigma_1 * \Sigma_2$	*iff*	$\exists h_1, h_2 \cdot\ h_1 \# h_2$ and $h{=}h_1{\cdot}h_2$, $s, h_1 \models \Sigma_1$ and $s, h_2 \models \Sigma_2$
$s, h \models \texttt{true}$	*iff*	always
$s, h \models \exists v.(\Pi \wedge \Sigma)$	*iff*	$\exists \alpha \cdot s[v{\mapsto}\alpha], h \models \Sigma$ and $s[v{\mapsto}\alpha] \models \Pi$
$s, h \models H_1 \vee H_2$	*iff*	$s, h \models H_1$ or $s, h \models H_2$

where $dom(f)$ is the domain of function f, $h_1 \# h_2$ denotes disjoint heaps h_1 and h_2 i.e., $dom(h_1){\cap}dom(h_2){=}\emptyset$, and $h_1{\cdot}h_2$ denotes the union of two disjoint heaps. If s is a stack, $v{\in}$*Var*, $v{\notin}dom(s)$ and $\alpha{\in}$*Values*$\cup$*Loc*, we write $s[v{\mapsto}\alpha] \equiv s{\cup}\{(v, \alpha)\}$. Note that in a concrete memory model e.g., RAM model, field names of points-to predicates are transformed into offsets. Then the pair $(f_i, s(E_i)$ (for all $i \in \{1...N\}$) is interpreted as $s(E){+}off_{f_i} = s(E_i)$ where off_{f_i} is the corresponding offset of field f_i. Entailment $H \models H'$ holds iff for all s and h, we have if $s, h \models H$ then $s, h \models H'$.

III. Proof Search

The core of the proposed bi-abductive procedure is a search algorithm that, given some starting entailment, scans through the set of proof rules for a rule that can be applied to the entailment. Once such a rule is identified, it is applied to the entailment to obtain some new sub-goals (and possibly fragments of the frame or anti-frame) until it can make a decision as to whether or not the entailment holds. All proof rules in our system are checked and applied in a specific order, aiming to ensure the process identifies the weakest possible preconditions and the strongest possible postconditions, while maintaining the validity of the solutions and forbidding trivial or `false` preconditions. The rules are searched in the following order:

1) Firstly, *Normalization* rules are applied exhaustively to unfold all possible scenarios and transform the antecedent of the input entailment into a normalised form.
2) Secondly, *Subtraction* rules are applied to match terms across the antecedent and consequent, removing them where possible. If heaps in the two sides are empty, the algorithm returns successfully.

3) Finally, it checks whether or not any *Bi-Abduction* rules could be applied to infer an anti-frame and frame. That is to identify portions of heap and pure information missing/immutable from the entailment, moving terms missing from the antecedent to the precondition and moving immutable terms from the antecedent to the postconditions. Following this, it repeats the first phase.

The normalization phase is essential for the subsequent two phases. It ensures that every spatial predicate in a normalized formula is precise; that is, given that $s, h \models H$ and H is in normal form, then for any spatial predicate $p \in H$, there exists *one and only one* sub-heap $h' \subseteq h$ such that $s, h' \models p$. In turn, this precision helps to guide the matching and inference rules, as given a predicate p on one side of a normalized entailment, there exists one and only one predicate p' on the other side such that p and p' are matched using the rules in the second phase. Furthermore, if we could not find such a predicate p' in the existing entailment, p' is eventually missing. The inference rules in the third phase reveal the missing predicates and then add them back to the entailment. This forms the foundation for the inference mechanism in our work.

IV. Inference Rules

We now discuss the general design of each of the rule-sets used in our proof search. A selection of these rules is presented in Figure 2 for reference[1]. Several of the rules present in our system include side conditions that must hold for the rule to be applied, even if it otherwise matches with the entailment. These side conditions enforce a number of constraints over the rules, preventing scenarios such as the inference of directly contradicting terms or the application of a rule over a state which does not fully support that application. These conditions can come in two forms: a discrete constraint preventing a rule from being applied if it is not met, or the introduction of a new entailment that must be satisfiable in order for the proof search to be completed.

A. Normalisation

Before the design of the normalization rules is discussed, we first present a normalised form that has a precise interpretation over its heap. For the normal form, we write $op(E)$ to denote either $E{\mapsto}[\rho]$, $ls(E, F)$ or $sls(E, V, V', F)$. Furthermore, the guard $G(op(E))$ is defined by $G(E{\mapsto}[\rho]) \stackrel{\text{def}}{=} \texttt{true}$, $G(ls(E, F)) \stackrel{\text{def}}{=} E{\neq}F$ and $G(sls(E, V, V', F)) \stackrel{\text{def}}{=} E{\neq}F$. The normal form itself is defined as follows:

Definition 1 (Normal Form): A formula $\Pi{\wedge}\Sigma$ is in normal form (NF for short) if:

1) $op(E) \in \Sigma$ implies $G(op(E)) \in \Pi$.
2) $op(E) \in \Sigma$ and $G(op(E)) \in \Pi$ imply $E{\neq}\texttt{null} \in \Pi$.
3) $op_1(E_1) * op_2(E_2) \in \Sigma$, $G(op_1(E_1)) \in \Pi$, and $G(op_2(E_2)) \in \Pi$ imply $E_1{\neq}E_2 \in \Pi$.
4) $E_1{=}E_2 \notin \Pi$.
5) $E{\neq}E \notin \Pi$.
6) Π is satisfiable.

If Δ is in NF and for any s and h such that $s, h \models \Delta$, $dom(h)$ is uniquely defined by s. A bi-abductive entailment is in NF if its LHS is in NF.

For each of the conditions required for a formula to be in normal form, there is an associated rule or rules designed to accomplish this soundly. Rule Node-EX presented in Figure 2 is one such example, designed to ensure that formulas meet condition 2 of the normal form, explicating the fact that given a heap h, for any $l \in dom(h)$, then $l \neq \texttt{null}$. Note that the side condition of the rule prevents the rule from being applied when there would be no effect.

B. Subtraction Rules

In this section, we present inference rules that match and subtract predicates on the right-hand side (RHS) against predicates in the LHS until both sides contain the empty heap emp. These rules are partitioned into three groups: decision rules when the heaps of both sides are emp; subtraction rules to match equivalent predicates on the two sides and generalization rules to match different (though similar) predicates from both sides. The first group contains two axiomatic rules EMP and IDENT as follows:

$$[\textbf{EMP}]$$
$$\frac{}{\Pi \wedge \texttt{emp} * [\texttt{true} \wedge \texttt{emp}] \vdash \texttt{true} \wedge \texttt{emp} * [\Pi \wedge \texttt{emp}]}$$

$$[\textbf{IDENT}]$$
$$\frac{}{\Delta * [\texttt{true} \wedge \texttt{emp}] \vdash \Delta * [\texttt{true} \wedge \texttt{emp}]}$$

That is, when both sides contain empty heaps, we copy the pure formula from the LHS as the inferred frame. Similarly, when two sides contain identical formula, the algorithm ends the search.

Rules in the second group either subtract pure formulas, split the entailment in order to subtract identical sub-formulas from both sides, or handle spatial predicates. The HYPOTHESIS rule is one such example from the second group, identifying a set of pure constraints in the consequent that is implied by the pure constraints in the antecedent and removing it from the entailment. The predicate-targeting rules aim to resolve entailments which feature overlapping fragments of spatial information, with rules such as LS-BASE specializing an inductive predicate by replacing it with its base case. Rules such as LS-REC(NODE) and SLS-REC(NODE) match and subtract the head node of an inductive predicate when it is matched with a node present in the antecedent, abducing necessary constraints in the process. These predicate-targeting rules are essentially unfolding operations over those shape predicates, explicating the segments that overlap in order to match and remove them from the entailment.

The final subset of subtraction rules are the generalization rules. These rules inductively generalize predicates in the LHS such that the matching between a predicate and a structurally similar predicate can be successful. As an example, in our

[1]While our system is more comprehensive than presented here, only the rules utilised in the examples in Section V are included due to spacing restrictions.

$$\frac{(G(op(E)) \wedge E{\neq}\texttt{null} \wedge \Delta * op(E)) * [M] \vdash \Delta' * [F]}{(G(op(E)) \wedge \Delta * op(E)) * [M] \vdash \Delta' * [F]} \text{ (where } E{\neq}\texttt{null}{\notin}\Delta\text{)} \quad [\textbf{NODE}{-}\textbf{EX}]$$

$$\frac{\Delta * [M] \vdash \Delta' * [F] \qquad \Delta * Q(E,E') \nvdash \texttt{false}}{\Delta * [M * Q(E,E')] \vdash \Delta' * Q(E,E') * [F]} \text{ (where } Q(E,E') \text{ is } op(E)\text{)} \quad [\textbf{INF}{-}\textbf{MISSING}]$$

$$\frac{\Pi \wedge \Sigma * [M] \vdash \Pi'' \wedge \Sigma' * [F]}{\Pi \wedge \Sigma * [M] \vdash \Pi' \wedge \Pi'' \wedge \Sigma' * [F]} \text{ (where } \Pi \Rightarrow \Pi'\text{)} \quad [\textbf{HYPOTHESIS}]$$

$$\frac{\Delta * [M] \vdash \Delta' * [F]}{\Delta * [M] \vdash \Delta' * ls(E,E) * [F]} \quad [\textbf{LS}{-}\textbf{BASE}]$$

$$\frac{\Delta * [M] \vdash \Delta' * ls(E_2,E_3) * [F]}{\Delta * E_1{\mapsto}[E_2] * [M] \vdash \Delta' * ls(E_1,E_3) * [F]} \text{ (where } E_1{\mapsto}[E_2] \notin \Delta'\text{)} \quad [\textbf{LS} - \textbf{REC}(\textbf{Node})]$$

$$\frac{V{\leq}V' \wedge \Delta * [M] \vdash \Delta' * ls(E_2,E_3) * [F]}{\Delta * sls(E_1,V,V',E_2) * [M] \vdash \Delta' * ls(E_1,E_3) * [F]} \text{ (where } sls(E,V,V',E') \notin \Delta'\text{)} \quad [\textbf{LS} - \textbf{GENERALIZE}]$$

$$\frac{\Delta * [M] \vdash \exists V_2.\ V_1{\leq}V_2 \wedge \Delta' * sls(E_2,V_2,V_3,E_3) * [F]}{\Delta * E_1{\mapsto}[E_2,V_1] * [M] \vdash \Delta' * sls(E_1,V_1,V_3,E_3) * [F]} \text{ (where } E_1{\mapsto}[E_2,V_1] \notin \Delta'\text{)} \quad [\textbf{SLS} - \textbf{REC}(\textbf{Node})]$$

Figure 2: Subset of Inference Rules

system, a sorted linked list indirectly entails a basic singly-linked list between the same points due to the fact that by our definition, all sorted lists are simple singly-linked lists at the structural level. This fact could be proved by inductively unfolding both predicates and eliminating the matching nodes. LS-GENERALIZE fires when a sorted list in the antecedent of the entailment shares a head node with a simple singly-linked list on the consequent, and essentially abstracts the ordering information away from the sorted list, leaving only the spatial information; a singly-linked list.

C. Bi-abduction Rules

When the search algorithm is stuck at Unfold-and-Match, our bi-abductive proof technique is invoked. The technique identifies some fragment of the anti-frame, adds it to the assumption on the antecedent and repeats the process of Unfold-and-Match with the now reduced entailment. Once the procedure is completed, the complete anti-frame is recovered by combining all of the separate fragments identified during each stage of the abduction process. The process of frame inference is performed similarly at the same time.

In order to maintain the validity of the sorted list predicates, rules targeting sls predicates take additional steps to ensure that the order of the sorted list is preserved throughout the application of the rules. In the case of SLS-REC (Node), this means the identification of a suitable ordering constraints between the sorted list and the eliminated node. Unlike the inequality between the two end-points, however, this ordering constraint is not abduced from the application of the rule, as it is implicit inside the sorted list predicates throughout. Instead, the ordering constraint is simply made explicit by the application of the rule. How this constraint is handled varies depending on the form of the rule; for SLS-REC (Node), the ordering constraint simply becomes an additional constraint in the consequent of the entailment, requiring proof that the ordering relation is satisfied in order to prove the overall entailment. The inference of these ordering constraints may be performed afterwards, if necessary.

While this may initially seem to be a useful aspect in ensuring the soundness of these rules, this identification of formerly implicit ordering constraints has an additional benefit. In previous works in the area of combined domain bi-abduction, the handling of complex pure terms, such as ordering, was typically undertaken partially or fully in a secondary analysis phase. By completely integrating the pure constraints into the abduction and entailment rules, we have developed a system in which the complex pure properties are identified *during* the shape analysis, eliminating the need for a subsequent pure analysis following this initial phase.

Should no other bi-abduction rule be applicable, INF-MISSING is applied, identifying the missing symbolic heap of the antecedent and adding it to the anti-frame (its framing counterpart, INF-EXTRA, though omitted, operates in a similar manner for the consequent and frame). While this is a typically necessary operation, bulk additions to the anti-frame may result in the identification of overly-complex specifications, should the previous rules not be applied. The greater precision of the earlier rules is therefore preferred. In these two rules, to avoid inference contradiction, the satisfiability of the solution is required. The solving of this satisfiability may be implemented through existing works like [11], [12], [13], [14], [15].

V. Working Examples

We now demonstrate how our technique would be applied to prove combined-domain entailments. The derivation trees for these examples may be found in Figure 3.

a) Example 1: We begin by demonstrating the system on a relatively straightforward example using sorted lists. The entailment used in this first example,

$$x{\mapsto}[n : x', v : a] * [?M] \vdash sls(x,a,b,null) * [?F]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x{\neq}\texttt{null} \wedge \texttt{emp} * [\texttt{emp} \wedge \texttt{true}] \vdash \texttt{true} \wedge \texttt{emp} * [x{\neq}\texttt{null}]}\,\textbf{[EMP]}}{x{\neq}\texttt{null} \wedge \texttt{emp} * [\texttt{emp} \wedge \texttt{true}] \vdash \exists a'.\ a{\leq}a' \wedge \texttt{emp} * [x{\neq}\texttt{null}]}\,\textbf{[HYPOTHESIS]}}{x{\neq}\texttt{null} \wedge \texttt{emp} * [\exists a'.sls(x', a', b, \texttt{null})] \vdash \exists a'.\ a{\leq}a' * sls(x', a', b, \texttt{null}) * [x{\neq}\texttt{null}]}\,\textbf{[INF-MISSING]}}{x{\neq}\texttt{null} \wedge x{\mapsto}[x', a] * [\exists a'.sls(x', a', b, \texttt{null})] \vdash sls(x, a, b, \texttt{null}) * [x{\neq}\texttt{null}]}\,\textbf{[SLS-REC (Node)]}}{x{\mapsto}[x', a] * [\exists a'.sls(x', a', b, \texttt{null})] \vdash sls(x, a, b, \texttt{null}) * [x{\neq}\texttt{null}]}\,\textbf{[NODE-EX]}$$

(a) Example 1.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{i{\leq}j \wedge \Pi \wedge \texttt{emp} * [\texttt{true} \wedge \texttt{emp}] \vdash \texttt{true} \wedge \texttt{emp} * [\Pi \wedge i{\leq}j]}\,\textbf{[EMP]}}{i{\leq}j \wedge \Pi \wedge \texttt{emp} * [\texttt{true} \wedge \texttt{emp}] \vdash ls(\texttt{null}, \texttt{null}) * [\Pi \wedge i{\leq}j]}\,\textbf{[LS-BASE]}}{\Pi \wedge sls(y, i, j, \texttt{null}) * [\texttt{true} \wedge \texttt{emp}] \vdash ls(y, \texttt{null}) * [\Pi \wedge i{\leq}j]}\,\textbf{[LS-GENERALIZE]}}{\Pi \wedge x{\mapsto}[y] * sls(y, i, j, \texttt{null}) * [\texttt{true} \wedge \texttt{emp}] \vdash ls(x, \texttt{null}) * [\Pi \wedge i{\leq}j]}\,\textbf{[LS-REC (Node)]}$$

(b) Example 2, where $\Pi \equiv x{\neq}\texttt{null} \wedge x \neq y \wedge y \neq \texttt{null}$.

Figure 3: Derivation Trees for Examples

is a relatively simple entailment that is likely to be encountered in some form across many example programs. In this case, for the entailment to be satisfied, the abduction procedure should identify x as the head of a sorted list and produce an anti-frame to reflect that.

Our algorithm first normalizes the entailment via the application of NODE-EX, identifying the non-null constraint over the node x. As no further normalization rules can be applied, the algorithm instead attempts to find an appropriate subtraction rule. SLS-REC(Node) is the first matching rule, identifying the overlap between node x in the antecedent and the head of the sls in the consequent and removing them. The application of this rule eliminates x from the entailment, selecting the "next" value of x as the new head and some fresh variable as the new minimum and identifying an ordering constraint between this new minimum and the previous one. At this point, no further normalisation or subtraction rules can be applied to the entailment, causing the algorithm to invoke the abduction rules. As the only viable rule, INF-MISSING is applied, identifying the missing sorted list and adding it to the anti-frame (The consistency check has been omitted for space). The system then identifies the existential ordering relation as a hypothesis and removes it, before finally, EMP is applied, ending the analysis and introducing the remaining pure information into the anti-frame.

b) Example 2: Consider the following entailment:

$$\Pi \wedge x{\mapsto}[n : y] * sls(y, mi, ma, \texttt{null}) * [?M] \vdash ls(x, \texttt{null}) * [?F]$$

In this example, the system should identify that the symbolic heap present on the left-hand side is already sufficient to describe a null-terminated singly-linked list pointed-to by x. The primary obstacle in this is the fact that the list segment pointed by y is represented as a sorted list, necessitating the use of a generalisation rule. The proof tree for this example may be found in Figure 3b; we omit the normalisation steps and compress the identified terms into Π for conciseness.

The system first normalises the entailment before proceeding to subtract node x from the ls predicate through the application of LS-REC (Node), leaving the ls and sls predicates the only remaining spatial formulas. An application of LS-GENERALIZE then abstracts the sorted list from the antecedent into a simple list which is then matched against the list in the consequent and removed, with the resulting $ls(\texttt{null}, \texttt{null})$ predicate the only remaining spatial term. This is subtracted via LS-BASE, reducing the entailment to pure constraints only. Finally, EMP is applied, adding the remaining pure constraints of the antecedent to the frame and ending the process. Important to note is the anti-frame returned by our system; as mentioned earlier, the existing state is sufficient to satisfy the entailment and this empty anti-frame accurately reflects this fact.

VI. Related Work

There are a number of works related to our technique. For classical bi-abduction, Infer [5] remains the most well-known and is essentially a "pure" implementation of the bi-abduction technique [5]. Extensions have been made to the tool, but it remains restricted to shape properties. Some extended versions of bi-abduction have also been investigated, with the second-order bi-abduction technique of Le *et. al.* [7], [16] being one of the most promising.

In the area of combined domain bi-abduction, there is a small number of relevant works. One of the earliest in this area is the work described in [8]. Building upon a set of shape properties obtained from some previous shape analysis, the technique introduces and extends the shape predicates with a set of pure terms representing properties such as size or order, alongside additional predicates representing relational information over the structure. A forwards analysis is then undertaken to generate proof obligations for the

relational predicates, which are then finalised via a fixpoint analysis, producing a precondition ensuring memory safety and termination. One of the most developed techniques for combined domain bi-abduction, and close to our own work, is the work of Qin et al [17]. This technique is also based around a fixpoint analysis of the target program, abducting necessary state components in each pass, developing stronger preconditions until a fixpoint is reached. This stage is guided through user-defined predicates outlining the expected data structures encountered, identifying several aspects of the pure domain in these passes. However, a secondary bi-abductive inference is still necessary to obtain smaller components of pure information during the final abstraction phase for each iteration, and widening operations utilised to accelerate reaching the fixpoint may introduce soundness issues. Nevertheless, the tool is quite effective, discovering specifications for data structures involving not only shape information, but size and bag properties as well. Since it is not truly a single-stage bi-abductive method as our system is, the performance of the tool is degraded by the two-phase analysis.

A range of other combined domain analysis techniques exist in the literature: `Thor` [9] is capable of handling pure properties alongside memory safety, able to operate over ordering, shape, size or depth properties via an additional analysis over a "proof program". Chang et al [18] describe a generalised framework for shape analysis based around forward abstract interpretation with additional support for relations between data values. The venerable TVLA system [19] is capable of supporting a wide range of structures and properties, including complex pure properties such as ordering, though is limited by the lack of support for compositional reasoning. Finally, techniques based on Forest Automata are also known [20], though few can operate in the combined domain [21].

VII. Conclusions and Future Work

We have presented a novel proof system for the bi-abduction problem in separation logic for lists with ordering properties. Our system has been designed based on the Unfold-and-Match paradigm: given an entailment, it systematically explores all candidate matching instances of the antecedent and consequent prior to inferring any missing portion. For future work, we will implement the proof system for automated program verification and may further extend the system with a more expressive fragment e.g., general inductive definitions with other pure properties (size, balance, bag/set). Another work would be bi-abduction to infer error specifications [22] for counterexample generation [23], [24] and program repair [25].

References

[1] H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn, "Scalable shape analysis for systems code," in *CAV*. Springer, 2008, pp. 385–398.

[2] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 48–62.

[3] J. C. Reynolds, "Separation logic: A logic for shared mutable data structures," in *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 2002, pp. 55–74.

[4] S. S. Ishtiaq and P. W. O'Hearn, "BI as an assertion language for mutable data structures," *ACM SIGPLAN Notices*, vol. 36, no. 3, pp. 14–26, 2001.

[5] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang, "Compositional shape analysis by means of bi-abduction," in *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 2009, pp. 289–300.

[6] C. Calcagno and D. Distefano, "Infer: An automatic program verifier for memory safety of c programs," in *NASA Formal Methods Symposium*. Springer, 2011, pp. 459–465.

[7] Q. L. Le, C. Gherghina, S. Qin, and W.-N. Chin, "Shape analysis via second-order bi-abduction," in *Computer Aided Verification*, A. Biere and R. Bloem, Eds. Cham: Springer International Publishing, 2014, pp. 52–68.

[8] M.-T. Trinh, Q. L. Le, C. David, and W.-N. Chin, "Bi-abduction with pure properties for specification inference," in *Asian Symposium on Programming Languages and Systems*, 2013, pp. 107–123.

[9] S. Magill, M.-H. Tsai, P. Lee, and Y.-K. Tsay, "Automatic numeric abstractions for heap-manipulating programs," in *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '10. ACM, 2010, pp. 211–222.

[10] J. Berdine, C. Calcagno, and P. W. Ohearn, "Symbolic execution with separation logic," in *Asian Symposium on Programming Languages and Systems*. Springer, 2005, pp. 52–68.

[11] Q. L. Le, M. Tatsuta, J. Sun, and W.-N. Chin, "A decidable fragment in separation logic withinductive predicates and arithmetic," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 495–517.

[12] Q. L. Le, J. Sun, and W.-N. Chin, "Satisfiability modulo heap-based programs," in *Computer Aided Verification*, S. Chaudhuri and A. Farzan, Eds. Cham: Springer International Publishing, 2016, pp. 382–404.

[13] M. Tatsuta, Q. L. Le, and W.-N. Chin, *Decision Procedure for Separation Logic with Inductive Definitions and Presburger Arithmetic*. Cham: Springer International Publishing, 2016, pp. 423–443.

[14] X. Gu, T. Chen, and Z. Wu, "A complete decision procedure for linearly compositional separation logic with data constraints," in *Automated Reasoning*, N. Olivetti and A. Tiwari, Eds. Cham: Springer International Publishing, 2016, pp. 532–549.

[15] Z. Xu, T. Chen, and Z. Wu, "Satisfiability of compositional separation logic with tree predicates and data constraints," in *Automated Deduction – CADE 26*, L. de Moura, Ed. Cham: Springer International Publishing, 2017, pp. 509–527.

[16] Q. L. Le, J. Sun, and S. Qin, "Frame inference for inductive entailment proofs in separation logic," in *Tools and Algorithms for the Construction and Analysis of Systems*, D. Beyer and M. Huisman, Eds. Cham: Springer International Publishing, 2018, pp. 41–60.

[17] S. Qin, G. He, W.-N. Chin, F. Craciun, M. He, and Z. Ming, "Automated specification inference in a combined domain via user-defined predicates," *Science of Computer Programming*, pp. 189–212, 2017.

[18] B.-Y. Chang and X. Rival, "Relational inductive shape analysis," in *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 2008, pp. 247–260.

[19] M. Sagiv, T. Reps, and R. Wilhelm, "Parametric shape analysis via 3-valued logic," *ACM Transactions on Programming Languages and Systems*, vol. 24, no. 3, pp. 217–298, 2002.

[20] L. Holík, M. Hruška, O. Lengál, A. Rogalewicz, J. Šimáček, and T. Vojnar, "Forester: Shape analysis using tree automata," in *TACAS*, C. Baier and C. Tinelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 432–435.

[21] P. A. Abdulla, L. Holík, B. Jonsson, O. Lengál, C. Q. Trinh, and T. Vojnar, "Verification of heap manipulating prorams with ordered data by extended forest automata," in *ATVA 2013, October 15-18, Hanoi, Vietnam*. Springer Berlin/Heidelberg, 2013, pp. 224–239.

[22] Q. L. Le, A. Sharma, F. Craciun, and W.-N. Chin, "Towards complete specifications with an error calculus," in *NASA Formal Methods*, G. Brat, N. Rungta, and A. Venet, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 291–306.

[23] L. H. Pham, Q. L. Le, Q.-S. Phan, and J. Sun, "Concolic testing heap-manipulating programs," in *FM 2019*, to appear.

[24] L. H. Pham, Q. L. Le, Q.-S. Phan, J. Sun, and S. Qin, "Enhancing symbolic execution of heap-based programs with separation logic for test input generation," in *ATVA 2019*, to appear.

[25] L. D. X. Bach, Q. L. Le, D. Lo, and C. L. Goues, "Enhancing automated program repair with deductive verification," in *2016 IEEE International Conference on Software Maintenance and Evolution*, 2016, pp. 428–432.