An Efficient Cyclic Entailment Procedure in a Fragment of Separation Logic

Quang Loc Le[1](✉) and Xuan-Bach D. Le[2]

[1] Department of Computer Science, University College London, London, UK
loc.le@ucl.ac.uk

[2] School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
bach.le@unimelb.edu.au

Abstract. An efficient entailment proof system is essential to compositional verification using separation logic. Unfortunately, existing decision procedures are either inexpressive or inefficient. For example, Smallfoot is an efficient procedure but only works with hardwired lists and trees. Other procedures that can support general inductive predicates run exponentially in time as their proof search requires back-tracking to deal with a disjunction in the consequent.
This paper presents a decision procedure to derive cyclic entailment proofs for general inductive predicates in polynomial time. Our procedure is efficient and does not require back-tracking; it uses normalisation rules that help avoid the introduction of disjunction in the consequent. Moreover, our decidable fragment is sufficiently expressive: It is based on compositional predicates and can capture a wide range of data structures, including sorted and nested list segments, skip lists with fast-forward pointers, and binary search trees. We implemented the proposal in a prototype tool, called $\mathtt{S2S}_{\mathtt{Lin}}$, and evaluated it over challenging problems from a recent separation logic competition. The experimental results confirm the efficiency of the proposed system.

Keywords: Cyclic Proofs, Entailment Procedure, Separation Logic.

1 Introduction

Separation logic [20,37] has successfully reasoned about programs manipulating pointer structures. It empowers reusability and scalability through compositional reasoning [6,7]. A compositional verification system relies on bi-abduction technology which is, in turn, based on entailment proof systems. Entailment is defined: Given an antecedent A and a consequent C where A and C are formulas in separation logic, the entailment problem checks whether $A \models C$ is valid. Thus, an efficient decision procedure for entailments is the vital ingredient of an automatic verification system in separation logic.

To enhance the expressiveness of the assertion language, for example, to specify unbounded heaps and interesting pure properties (e.g., sortedness, parent pointers), separation logic is typically combined with user-defined inductive predicates [9,31,35]. In this setting, one key challenge of an entailment procedure is the ability to support induction reasoning over the combination of heaps and data content. The problem of induction is challenging, especially for an automated inductive theorem prover, where the induction rules are not explicitly stated. Indeed, this problem is undecidable [1].

Developing a sound and complete entailment procedure that could be used for compositional reasoning is not trivial. It is unknown how model-based systems, e.g. [14,15,17,18,22,23], could support compositional reasoning. In contrast, there was evidence that proof-based decision procedures, e.g., Smallfoot [2] and the variant [12], and Cycomp [42], can be extended to solve the bi-abduction problem, which enables compositional reasoning and scalability [7,25]. Smallfoot was the centre of the biabductive procedure deployed in Infer [7], which which greatly impacted academia and industry [13]. Furthermore, Smallfoot is very efficient due to its use of the "exclude-the-middle" rule, which can avoid the proof search over the disjunction in the consequent. However, Smallfoot works for hardwired lists and binary trees only. In contrast, Cycomp, a recent complete entailment procedure, is a cyclic proof system without "exclude-the-middle" and can support general inductive predicates but has double exponential time complexity due to the proof search (and back-tracking) in the consequent.

This paper introduces a cyclic proof system with an "exclude-the-middle"-styled decision procedure for decidable yet expressive inductive predicates. We especially show that our procedure runs in polynomial time when the maximum number of fields of data structures is bounded by a constant. The decidable fragment, `SHLIDe`, contains inductive definitions of compositional predicates and pure properties. These predicates can capture nested list segments, skip lists and trees. The pure properties of small models can model a wide range of common data structures, e.g. a list with fast-forward pointers, sorted nested lists, and binary search trees [22,32]. This fragment is much more expressive than Smallfoot's and is incomparable to Cycomp's [42]: there exist some entailments our system can handle, but Cyccomp could not, and vice versa.

Our procedure is a variant of the cyclic proof system introduced by Brotherston [3,5] and has become one of the leading solutions to induction reasoning in separation logic. Intuitively, a cyclic proof is naturally represented as a tree of statements (entailments in this paper). The leaves are either axioms or nodes linked back to inner nodes; the tree's root is the theorem to be proven, and nodes are connected to one or more children by proof rules. Alternatively, a cyclic proof can be viewed as a tree possibly containing some back-links (a.k.a. cycles, e.g., "C, if B, if C") such that the proof satisfies some global soundness condition. This condition ensures that the proof can be viewed as a proof of *infinite descent*. For instance, for a cyclic entailment proof with inductive definitions, if every cycle contains an unfolding of some inductive predicate, then that predicate is infinitely often reduced into a strictly "smaller" predicate. This infinity is impossible as the semantics of inductive definitions only allows finite steps of unfolding. Hence, that proof path with the cycle can be disregarded.

The proposed system advances Brotherston's system in three ways. First, the proposed proof search algorithm is specialized to `SHLIDe`, which includes "exclude-the-middle" rules and excludes any back-tracking. The existing proof procedures typically search for proof (and back-track) over disjunctive cases generated from unfolding inductive predicates in the RHS of an entailment. To avoid such costly searches, we propose "exclude-the-middle"-styled normalised rules in which the unfolding of inductive predicates in the RHS always produces one disjunct. Therefore, our system is much more efficient than existing systems. Second, while a standard Brotherston system is incomplete, our proof search is complete in `SHLIDe`: If it is stuck (i.e., it can not apply any inference rules), then the root entailment is invalid.

Lastly, while the global soundness in [5] must be checked globally and explicitly, every back-link generated in `SHLIDe` is sound by design. We note that Cycomp, introduced in [42], was the first work to show the completeness of a cyclic proof system. However, in contrast to ours, it did not discuss the global soundness condition, which is the crucial idea attributing to the soundness of cyclic proofs.

Contributions Our primary contributions are summarized as follows.

- We present a novel decision procedure, $\texttt{S2S}_{\texttt{Lin}}$, for the entailment problem in separation logic with inductive definitions of compositional predicates.
- We provide a complexity analysis of the procedure.
- We have implemented the proposal in a prototype tool and tested it with the SL-COMP benchmarks [38,39]. The experimental results show that $\texttt{S2S}_{\texttt{Lin}}$ is effective and efficient compared to state-of-the-art solvers.

Organization The remainder of the paper is organised as follows. Sect. 2 describes the syntax of formulas in fragment `SHLIDe`. Sect. 3 presents the basics of an "exclude-the-middle" proof system and cyclic proofs. Sect. 4 elaborates on the result, the novel cyclic proof system, including an illustrative example. Sect. 5 discusses soundness and completeness. Sect. 6 presents the implementation and evaluation. Sect. 7 discusses related work. Finally, Sect. 8 concludes the work.

2 Decidable Fragment `SHLIDe`

Subsection 2.1 presents syntax of separation logic formulae and recursive definitions of linear predicates and local properties. Subsection 2.2 shows semantics.

2.1 Separation Logic Formulas

Concrete heap models assume a fixed finite collection of data structures *Node*, a fixed finite collection of field names *Fields*, a set *Loc* of locations (heap addresses), a set of non-addressable values *Val*, with the requirement that $\mathit{Val}{\cap}\mathit{Loc}{=}\emptyset$ (i.e., no pointer arithmetic). `null` is a special element of *Val*. $\mathbb{Z}$ denotes the set of integers ($\mathbb{Z}{\subseteq}\mathit{Val}$) and k denotes integer numbers. *Var* an infinite set of variables, $\bar{v}$ a sequence of variables.

Syntax Disjunctive formula Φ, symbolic heaps Δ, spatial formula κ, pure formula π, pointer (dis)equality ϕ, and (in)equality formula α are as follows.

$$\Phi ::= \Delta \mid \Phi \vee \Phi \qquad \Delta ::= \kappa{\wedge}\pi \mid \exists v.\ \kappa{\wedge}\pi \qquad \pi ::= \texttt{true} \mid \alpha \mid \neg\pi \mid \pi{\wedge}\pi$$
$$\kappa ::= \texttt{emp} \mid x{\mapsto}c(f{:}v, .., f{:}v) \mid \texttt{P}(\bar{v}) \mid \kappa{*}\kappa \qquad \alpha ::= a{=}a \mid a{\leq}a \qquad a ::= k \mid v$$

where $v{\in}\mathit{Var}$, $c{\in}\mathit{Node}$ and $f{\in}\mathit{Fields}$. Note that we often discard field names f of points-to predicates $x{\mapsto}c(f{:}v, .., f{:}v)$ and use the short form as $x{\mapsto}c(\bar{v})$. $v_1{\neq}v_2$ is the short form of $\neg(v_1{=}v_2)$. E denotes for either a variable or `null`. $\Delta[E/v]$ denotes the formula obtained from Δ by substituting v by E. *A symbolic heap is referred as a base, denoted as* Δ^b, *if it does not contain any occurrence of inductive predicates.*

Inductive Definitions We write $\mathcal{P}$ to denote a set of n defined predicates $\mathcal{P}{=}\{\mathtt{P_1}, ..., \mathtt{P_n}\}$ in our system. Each inductive predicate has following types of parameters: a pair of root and segment defining segment-based linked points-to heaps, reference parameters (e.g., parent pointers, fast-forwarding pointers), transitivity parameters (e.g., singly-linked lists where every heap cell contains the same value a) and pairs of ordering parameters (e.g., trees being binary search trees). An inductive predicate is defined as

$$\begin{array}{l}\mathtt{pred}\ \mathtt{P}(r,F,\bar{B},u,sc,tg) \equiv \mathtt{emp}{\wedge}r{=}F{\wedge}sc{=}tg \\ \qquad \vee\ \exists X_{tl}, \bar{Z}, sc'.r{\mapsto}c(X_{tl},\bar{p},u,sc') * \kappa' * \mathtt{P}(X_{tl},F,\bar{B},u,sc',tg) \wedge r{\neq}F \wedge sc \diamond sc'\end{array}$$

where r is the root, F the segment, $\bar{B}$ the borders, u the parameter for a transitivity property, sc and tg source and target, respectively, parameters of an order property, $r{\mapsto}c(X_{tl},\bar{p},u,sc') * \kappa'$ the matrix of the heaps, and $\diamond \in \{=, \geq, \leq\}$. (The extension for multiple local properties is straightforward.) Moreover, this definition is constrained by the following three conditions on heap connectivity, establishment, and termination.

Condition C1. In the recursive rule, $\bar{p} = \{\mathtt{null}\}{\cup}\bar{Z}$. This condition implies that If two variables points to the same heap, their content must be the same. For instance, the following definition of singly-linked lists of even length does not satisfy this condition.

$$\mathtt{pred}\ \mathtt{ell}(r,F) \equiv \mathtt{emp}{\wedge}r{=}F\ \vee\ \exists x_1,X.r{\mapsto}c_1(x_1){*}x_1{\mapsto}c_1(X){*}\mathtt{ell}(X,F){\wedge}r{\neq}F$$

as n_3 and X are not field variables of the node pointed-to by r.

Condition C2. The matrix heap defines nested and connected list segments as:

$$\kappa'{:=}\mathtt{Q}(Z,\bar{U})\ |\ \kappa'{*}\kappa'\ |\ \mathtt{emp}$$

where $Z{\in}\bar{p}$ and $(\bar{U} \setminus \bar{p}) \cap Z = \emptyset$. This condition ensures connectivity (i.e. all allocated heaps are connected to the root) and establishment (i.e. every existential quantifier either is allocated or equals to a parameter).

Condition C3. There is no mutual recursion. We define an order $\prec_{\mathcal{P}}$ on inductive predicates as: $\mathtt{P} \prec_{\mathcal{P}} \mathtt{Q}$ if at least one occurrence of predicate $\mathtt{Q}$ appears in the definition of $\mathtt{P}$ and $\mathtt{Q}$ is called a direct sub-term of $\mathtt{P}$. We use $\prec^*_{\mathcal{P}}$ to denote the transitive closure of $\prec_{\mathcal{P}}$.

Several definition examples are shown as follows.

$$\begin{array}{l}\mathtt{pred}\ \mathtt{ll}(r,F) \equiv \mathtt{emp}{\wedge}r{=}F\ \vee\ \exists X_{tl}.r{\mapsto}c_1(X_{tl}){*}\mathtt{ll}(X_{tl}, F){\wedge}r{\neq}F \\ \mathtt{pred}\ \mathtt{nll}(r,F,B) \equiv \mathtt{emp}{\wedge}r{=}F \\ \quad \vee\ \exists X_{tl},Z.r{\mapsto}c_3(X_{tl},Z){*}\mathtt{ll}(Z, B){*}\mathtt{nll}(X_{tl},F,B){\wedge}r{\neq}F \\ \mathtt{pred}\ \mathtt{skl1}(r,F) \equiv \mathtt{emp}{\wedge}r{=}F\ \vee\ \exists X_{tl}.r{\mapsto}c_4(X_{tl},\mathtt{null},\mathtt{null}){*}\mathtt{skl1}(X_{tl}, F){\wedge}r{\neq}F \\ \mathtt{pred}\ \mathtt{skl2}(r,F) \equiv \mathtt{emp}{\wedge}r{=}F \\ \quad \vee\ \exists X_{tl}, Z_1.r{\mapsto}c_4(Z_1,X_{tl},\mathtt{null}){*}\mathtt{skl1}(Z_1,X_{tl}){*}\mathtt{skl2}(X_{tl}, F){\wedge}r{\neq}F \\ \mathtt{pred}\ \mathtt{skl3}(r,F) \equiv \mathtt{emp}{\wedge}r{=}F \\ \quad \vee\ \exists X_{tl},Z_1,Z_2.r{\mapsto}c_4(Z_1,Z_2,X_{tl}){*}\mathtt{skl1}(Z_1,Z_2){*}\mathtt{skl2}(Z_2,X_{tl}){*}\mathtt{skl3}(X_{tl},F){\wedge}r{\neq}F \\ \mathtt{pred}\ \mathtt{tree}(r,B) \equiv \mathtt{emp}{\wedge}r{=}B \\ \quad \vee\ \exists r_l, r_r.r{\mapsto}c_t(r_l,r_r){*}\mathtt{tree}(r_l,B){*}\mathtt{tree}(r_r,B) \wedge r{\neq}B\end{array}$$

`ll` defines singly-linked lists, `nll` defines lists of acyclic lists, `slk1`, `slk2` and `slk3` define skip-lists. Finally, `tree` defines binary trees. We extend predicate `ll` with transitivity and order parameters to obtain predicate `lla` and `lls`, respectively, as follows.

$$\begin{array}{l}\texttt{pred lla}(r,F,a) \equiv \texttt{emp}\wedge r{=}F \ \vee\ \exists X_{tl}.r{\mapsto}c_2(X_{tl},a) * \texttt{lla}(X_{tl},F,a)\wedge r{\neq}F\\ \texttt{pred lls}(r,F,mi,ma) \equiv \texttt{emp}\wedge r{=}F\wedge ma{=}mi\\ \quad \vee\ \exists X_{tl},mi_1.r{\mapsto}c_4(X_{tl},mi_1) * \texttt{lls}(X_{tl},F,mi_1,ma)\wedge r{\neq}F \wedge mi{\leq}mi_1\end{array}$$

Unfolding Given $\texttt{pred P}(\bar{t}) \equiv \Phi$ and a formula $\texttt{P}(\bar{v}){*}\Delta$, then unfolding $\texttt{P}(\bar{v})$ means replacing $\texttt{P}(\bar{v})$ by $\Phi[\bar{v}/\bar{t}]$. We annotate a number, called unfolding number, for each occurrence of inductive predicates. Suppose $\exists\bar{w}.r{\mapsto}c(\bar{p}) * \texttt{Q}_1(\bar{v}_1){*}...{*}\texttt{Q}_m(\bar{v}_m) * \texttt{P}(\bar{v}_0){\wedge}\pi$ be the recursive rule, then in the unfolded formula, if $\texttt{P}(\bar{v}_0[\bar{v}/\bar{t}])^{k_1}$ and $\texttt{Q}_\texttt{i}(...)^{k_2}$ are direct sub-terms of $\texttt{P}(\bar{v})^k$ like above, then $k_1{=}k{+}1$ and $k_2 = 0$. When it is unambiguous, we discard the annotation of the unfolding number for simplicity.

2.2 Semantics

The program state is interpreted by a pair (s,h) where $s{\in}$*Stacks*, $h{\in}$*Heaps* and stack *Stacks* and heap *Heaps* are defined as:

$$\begin{array}{l}\mathit{Heaps} \overset{\text{def}}{=} \mathit{Loc}{\rightharpoonup}_{fin}(\mathit{Node} \rightarrow (\mathit{Fields} \rightarrow \mathit{Val} \cup \mathit{Loc})^m)\\ \mathit{Stacks} \overset{\text{def}}{=} \mathit{Var} \rightarrow \mathit{Val} \cup \mathit{Loc}\end{array}$$

Note that we assume that every data structure contains at most m fields. Given a formula Φ, its semantics is given by a relation: $s,h \models \Phi$ in which the stack s and the heap h satisfy the constraint Φ. The semantics is shown below

$$\begin{array}{lll}
s,h \models \texttt{emp} & \texttt{iff} & \mathit{dom}(h){=}\emptyset\\
s,h \models v{\mapsto}c(f_i : v_i) & \texttt{iff} & \mathit{dom}(h){=}\{s(v)\}, h(s(v)){=}g, g(c,f_i){=}s(v_i)\\
s,h \models P(\bar{v}) & \texttt{iff} & (h, s(\bar{v}_1), .., s(\bar{v}_k)) \in [\![P]\!]\\
s,h \models \kappa_1 * \kappa_2 & \texttt{iff} & \exists h_1,h_2\ s.t\ h_1\#h_2,\ h{=}h_1{\cdot}h_2,,\ s,h_1 \models \kappa_1 \text{ and } s,h_2 \models \kappa_2\\
s,h \models \texttt{true} & \texttt{iff} & \text{always}\\
s,h \models \kappa{\wedge}\pi & \texttt{iff} & s,h \models \kappa \text{ and } s \models \pi\\
s,h \models \exists v.\Delta & \texttt{iff} & \exists\alpha.s[v{\mapsto}\alpha],h \models \Delta\\
s,h \models \Phi_1 \vee \Phi_2 & \texttt{iff} & s,h \models \Phi_1 \text{ or } s,h \models \Phi_2
\end{array}$$

$\mathit{dom}(g)$ is the domain of g, $h_1\#h_2$ denotes disjoint heaps h_1 and h_2 i.e., $\mathit{dom}(h_1){\cap}\mathit{dom}(h_2){=}\emptyset$, and $h_1{\cdot}h_2$ denotes the union of two disjoint heaps. If s is a stack, $v{\in}\mathit{Var}$, and $\alpha{\in}\mathit{Val}{\cup}\mathit{Loc}$, we write $s[v{\mapsto}\alpha] = s$ if $v{\in}\mathit{dom}(s)$, otherwise $s[v{\mapsto}\alpha] = s{\cup}\{(v,\alpha)\}$. Semantics of non-heap (pure) formulas is omitted for simplicity. The interpretation of an inductive predicate $\texttt{P}(\bar{t})$ is based on the least fixed point semantics $[\![\texttt{P}]\!]$.

Entailment $\Delta \models \Delta'$ holds iff for all s and h, if $s,h \models \Delta$ then $s,h \models \Delta'$.

3 Entailment Problem & Overview

Throughout this work, we consider the following problem.

PROBLEM:	$\texttt{QF\_ENT}{-}\texttt{SL}_{\texttt{LIN}}$.
INPUT:	$\Delta_a \equiv \kappa_a{\wedge}\pi_a$ and $\Delta_c \equiv \kappa_c{\wedge}\pi_c$ where $FV(\Delta_c) \subseteq FV(\Delta_a) \cup \{\texttt{null}\}$.
QUESTION:	Does $\Delta_a \models \Delta_c$ hold?

An entailment, denoted as $\mathtt{e}$, is syntactically formalized as: $\Delta_a \vdash \Delta_c$ where Δ_a and Δ_c are quantifier-free formulas whose syntax are defined in the preceding section.

In Sect. 3.1, we present the basis of an exclude-the-middle proof system and our approach to $\mathtt{QF\_ENT{-}SL_{LIN}}$. In Sect. 3.2, we describe the foundation of cyclic proofs.

3.1 Exclude-the-Middle Proof System

Given a goal $\Delta_a \vdash \Delta_c$, an entailment proof system might derive entailments with a disjunction in the right-hand side (RHS). Such an entailment can be obtained by a proof rule that replaces an inductive predicate by its definition rules. Authors of Smallfoot [2] introduced a normal form and proof rules to prevent such entailments when the predicate are lists or trees. Smallfoot considers the following two scenarios.

- **Case 1** (Exclude-the-middle and Frame): The inductive predicate matches with a points-to predicate in the left-hand side (LHS). For instance, let us consider an entailment which is of the form $\mathtt{e}_1 : x{\mapsto}c(z) * \Delta \vdash \mathtt{ll}(x,y) * \Delta'$, where $\mathtt{ll}$ is singly-linked lists and $\mathtt{ll}(x,y)$ matches with $x{\mapsto}c(z)$ as they have the same root x. A typical proof system might search for proof through two definition rules of predicate $\mathtt{ll}$ (i.e., by unfolding $\mathtt{ll}(x,y)$ into two disjuncts): One includes the base case with $x = y$, and another contains the recursive case with $x \neq y$. Smallfoot prevents such unfolding by excluding the middle in the LHS: It reduces the entailment into two premises: $x{\mapsto}c(z) * \Delta \wedge x = y \vdash \mathtt{ll}(x,y) * \Delta'$ and $x{\mapsto}c(z) * \Delta \wedge x \neq y \vdash \mathtt{ll}(x,y) * \Delta'$. The first one considers the base case of the list (that is, $\mathtt{ll}(x,x)$) and is equivalent to $x{\mapsto}c(z) * \Delta \wedge x = y \vdash \Delta'$. Furthermore, the second premise checks the inductive case of the list and is equivalent to $\Delta \wedge x \neq y \vdash \mathtt{ll}(x,z) * \Delta'$.
- **Case 2** (Induction proving via hard-wired Lemma). The inductive predicate matches other inductive predicates in the LHS. For example, consider the entailment $\mathtt{e}_2 : \mathtt{ll}(x,z) * \Delta \vdash \mathtt{ll}(x,\mathtt{null}) * \Delta'$. Smallfoot handle $\mathtt{e}_2$ by using a proof rule as the consequence of applying the following hard-wired lemma $\mathtt{ll}(x,z) * \mathtt{ll}(z,\mathtt{null}) \models \mathtt{ll}(x,\mathtt{null})$ and reduces the entailment to $\Delta \vdash \mathtt{ll}(z,\mathtt{null}) * \Delta'$.

In doing so, Smallfoot does not introduce a disjunction in the RHS. However, as it uses specific lemmas in the induction reasoning, it only works for the hardwired lists.

This paper proposes $\mathtt{S2S_{Lin}}$ as an exclude-the-middle system for user-defined predicates, those in $\mathtt{SHLIDe}$. Instead of using hardwired lemmas, we apply cyclic proofs for induction reasoning. For instance, to discharge the entailment $\mathtt{e}_2$ above, $\mathtt{S2S_{Lin}}$ first unfolds $\mathtt{ll}(x,z)$ in the LHS and obtains two premises:

- $\mathtt{e}_{21} : (\mathtt{emp} \wedge x = z) * \Delta \vdash \mathtt{ll}(x,\mathtt{null}) * \Delta'$; and
- $\mathtt{e}_{22} : (x{\mapsto}c(y) * \mathtt{ll}(y,z) \wedge x \neq z) * \Delta \vdash \mathtt{ll}(x,\mathtt{null}) * \Delta'$

While it reduces $\mathtt{e}_{21}$ to $\Delta[z/x] \vdash \mathtt{ll}(z,\mathtt{null}) * \Delta'[z/x]$, for $\mathtt{e}_{22}$, it further applies the frame rule as in **Case 1** above and obtains $\mathtt{ll}(y,z) * \Delta \wedge x \neq z \vdash \mathtt{ll}(y,\mathtt{null}) * \Delta'$. Then, it makes a backlink between the latter and $\mathtt{e}_2$ and closes this path. Doing so does not introduce disjunctions in the RHS and can handle user-defined predicates.

3.2 Cyclic Proofs

Central to our work is a procedure that constructs a cyclic proof for an entailment. Given an entailment $\Delta \vdash \Delta'$, if our system can derive a cyclic proof, then $\Delta \models \Delta'$. If instead, it is stuck without proof, then $\Delta \models \Delta'$ is not valid.

The procedure includes proof rules, each of which is of the form:

$$\texttt{PR}_0 \; \frac{\texttt{e}_1 \quad \ldots \quad \texttt{e}_n}{\texttt{e}} \; \texttt{cond}$$

where entailment $\texttt{e}$ (called the conclusion) is reduced to entailments $\texttt{e}_1$, ..,$\texttt{e}_n$ (called the premises) through inference rule $\texttt{PR}_0$ given that the *side condition* $\texttt{cond}$ holds.

A cyclic proof is a proof tree $\mathcal{T}_i$ which is a tuple $(V, E, \mathcal{C})$ where

- V is a finite set of nodes representing entailments derived during the proof search;
- A directed edge $(\texttt{e}, \texttt{PR}, \texttt{e}') \in E$ (where $\texttt{e}'$ is a child of $\texttt{e}$) means that the premise $\texttt{e}'$ is derived from the conclusion $\texttt{e}$ via inference rule $\texttt{PR}$. For instance, suppose that the rule $\texttt{PR}_0$ above has been applied, then the following n edges are generated: $(\texttt{e}, \texttt{PR}_0, \texttt{e}_1)$, .., $(\texttt{e}, \texttt{PR}_0, \texttt{e}_n)$;
- and $\mathcal{C}$ is a partial relation which captures back-links in the proof tree. If $\mathcal{C}(\texttt{e}_c {\rightarrow} \texttt{e}_b, \sigma)$ holds, then $\texttt{e}_b$ is linked back to its ancestor $\texttt{e}_c$ through the substitution σ (where $\texttt{e}_b$ is referred to as a *bud* and $\texttt{e}_c$ is referred to as a *companion*). In particular, $\texttt{e}_c$ is of the form: $\Delta \vdash \Delta'$ and $\texttt{e}_b$ is of the form: $\Delta_1 {\wedge} \pi \vdash \Delta'_1$ where $\Delta \equiv \Delta_1\sigma$ and $\Delta' \equiv \Delta'_1\sigma$.

A leaf node is marked as closed if it is evaluated as valid (i.e. the node is applied with an axiom), invalid (i.e. no rule can apply), or linked back. Otherwise, it is marked as open. A proof tree is *invalid* if it contains at least one invalid leaf node. It is *pre-proof* if all its leaf nodes are either valid or linked back. Furthermore, a pre-proof is a cyclic proof if a global soundness condition is established in the tree. Intuitively, this condition requires that for every $\mathcal{C}(\texttt{e}_c {\rightarrow} \texttt{e}_b, \sigma)$, there exist inductive predicates $\texttt{P}(\bar{t_1})$ in $\texttt{e}_c$ and $\texttt{Q}(\bar{t_2})$ in $\texttt{e}_b$ such that $\texttt{Q}(\bar{t_2})$ is a subterm of $\texttt{P}(\bar{t_1})$.

Definition 1 (Trace) *Let $\mathcal{T}_i$ be a pre-proof of $\Delta_a \vdash \Delta_c$ and $(\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq 0}$ be a path of $\mathcal{T}_i$. A trace following $(\Delta_{a_i} {\vdash} \Delta_{c_i})_{i \geq 0}$ is a sequence $(\alpha_i)_{i \geq 0}$ such that each α_i (for all $i {\geq} 0$) is a subformula of Δ_{a_i} containing predicate $\texttt{P}(\bar{t})^u$, and either:*

- *α_{i+1} is the subformula occurrence in $\Delta_{a_{i+1}}$ corresponding to α_i in Δ_{a_i}.*
- *or $\Delta_{a_i} \vdash \Delta_{c_i}$ is the conclusion of a left-unfolding rule, $\alpha_i \equiv \texttt{P}(\bar{t})^u$ is unfolded, and α_{i+1} is a subformula in $\Delta_{a_{i+1}}$ and is the definition rule of $\texttt{P}(\bar{x})^u[\bar{t}/\bar{x}]$. In this case, i is said to be a* progressing point *of the trace.*

Definition 2 (Cyclic proof) *A pre-proof $\mathcal{T}_i$ of $\Delta_a \vdash \Delta_c$ is a cyclic proof if, for every infinite path $(\Delta_{a_i} {\vdash} \Delta_{c_i})_{i \geq 0}$ of $\mathcal{T}_i$, there is a tail of the path $p{=}(\Delta_{a_i} \vdash \Delta_{c_i})_{i \geq n}$ such that there is a trace following p which has infinitely progressing points.*

Suppose that all proof rules are (locally) sound (i.e., if the premises are valid, then the conclusion is valid). The following Theorem shows *global soundness*.

Theorem 1 (Soundness [5]). *If there is a cyclic proof of $\Delta_a \vdash \Delta_c$, then $\Delta_a \models \Delta_c$.*

The proof is by contraction (c.f. [5]). Intuitively, if we can derive a cyclic proof for $\Delta_a \vdash \Delta_c$ and $\Delta_a \not\models \Delta_c$, then the inductive predicates at the progress points are unfolded infinitely often. This infinity contradicts the least semantics of the predicates.

4 Cyclic Entailment Procedure

This section presents our main proposal, the entailment procedure ω-ENT with the proposed inference rules (subsection 4.1), and an illustrative example (subsection 4.2).

4.1 Proof Search

The proof search algorithm ω-ENT is presented in Fig. 1. ω-ENT takes $\texttt{e}_0$ as input, produces cyclic proofs, and based on that, decides whether the input is `valid` or `invalid`. The idea of ω-ENT is to iteratively reduce $\mathcal{T}_0$ into a sequence of cyclic proof trees $\mathcal{T}_i$, $i \geq 0$. Initially, for every $\texttt{P}(\bar{v})^k \in \texttt{e}_0$, k is reset to 0, and $\mathcal{T}_0$ only has $\texttt{e}_0$ as an open leaf, the root. On line 3, through the procedure $\texttt{is\_closed}(\mathcal{T}_i)$, ω-ENT chooses an *open* leaf node $\texttt{e}_i$, and a proof rule PR_i to apply. If $\texttt{is\_closed}(\mathcal{T}_i)$ returns `valid` (that is, every leaf is applied to an axiom rule or involved in a back-link), ω-ENT returns `valid` on line 4. If it returns `invalid`, then ω-ENT returns `invalid` (one line 5). Otherwise, it tries to link $\texttt{e}_i$ back to an internal node (on line 6). If this attempt fails, it applies the rule (line 7).

```
ω−ENT
input: e_0                    output: valid or invalid
1: $i \leftarrow 0$; $\mathcal{T}_i \leftarrow \texttt{e}_0$;
2: while true do
3:    $(\texttt{res}, \texttt{e}_i, PR_i) \leftarrow \texttt{is\_closed}(\mathcal{T}_i)$;
4:    if res=valid then return valid;
5:    if res=invalid then return invalid;
6:    if $\texttt{link\_back}_{\texttt{e}}(\mathcal{T}_i, \texttt{e}_i) = \texttt{false}$ then
7:       $\mathcal{T}_{i+1} \leftarrow \texttt{apply}(\mathcal{T}_i, \texttt{e}_i, PR_i)$;
8:    $i \leftarrow i+1$;
9: end
```

Fig. 1: Proof tree construction procedure

Note that at each leaf, `is_closed` attempts rules in the following order: normalization rules, axiom rules, and reduction rules. A rule PR_i is chosen if its conclusion can be unified with the leaf through some substitution σ. Then, on line 7, for each premise of PR_i, procedure `apply` creates a new open node and connects the node to $\texttt{e}_i$ via a new edge. If PR_i is an axiom, procedure `apply` marks $\texttt{e}_i$ as closed and returns.

Procedure $\texttt{is\_closed}(\mathcal{T}_i)$ This procedure examines the following three cases.

1. First, if all leaf nodes are marked closed, and none is `invalid`, then `is_closed` returns `valid`.
2. Secondly, `is_closed` returns `invalid` if there exists an open leaf node $\texttt{e}_i : \Delta \vdash \Delta'$ in NF such that one of the four following conditions hold:
   (a) $\texttt{e}_i$ could not be applied by any inference rule.
   (b) there exists a predicate $op_1(E) \in \Delta$ such that $op_2(E) \notin \Delta'$ and one of the following conditions holds:
      – either $\texttt{P}(E', E, ...)$ or $E' {\mapsto} c(E, ..)$ are on both sides
      – both $\texttt{P}(E', E, ...) \notin \Delta$ and $E' {\mapsto} c(E, ..) \notin \Delta$
   (c) there exists a predicate $op_1(E) {\in} \Delta'$ such that $G(op_1(E)) {\in} \Delta$ and $op_2(E) {\notin} \Delta$.
   (d) there exist $x {\mapsto} c_1(\bar{v}_1) \in \Delta$, $x {\mapsto} c_2(\bar{v}_2) \in \Delta'$ such that $c_1 \not\equiv c_2$ or $\bar{v}_1 {\not\equiv} \bar{v}_2$.
3. Lastly, an open leaf node $\texttt{e}_i$ could be applied by an inference rule (e.g. PR_i), `is_closed` returns the triple (`unknown`, $\texttt{e}_i$, PR_i).

In the rest, we discuss the proof rules and the auxiliary procedures in detail.

Normalisation An entailment is in the normal form (NF) if its LHS is in NF. We write $op(E)$ to denote for either $E{\mapsto}c(\bar{v})$ or $\mathtt{P}(E,F,\bar{B},\bar{v})$. Furthermore, the guard $G(op(E))$ is defined by: $G(E{\mapsto}c(\bar{v})) \overset{\text{def}}{=} \mathtt{true}$ and $G(\mathtt{P}(E,F,\bar{B},\bar{v})) \overset{\text{def}}{=} E{\neq}F$.

Definition 3 (Normal Form) *A formula $\kappa{\wedge}\phi{\wedge}a$ is in normal form if:*

1. $op(E) \in \kappa$ *implies* $G(op(E)) \in \phi$
2. $op(E) \in \kappa$ *implies* $E{\neq}\mathtt{null} \in \phi$
3. $op_1(E_1) * op_2(E_2) \in \kappa$ *implies* $E_1{\neq}E_2 \in \phi$
4. $E_1{=}E_2 \notin \phi$
5. $E{\neq}E \notin \phi$
6. *a is satisfiable*

If Δ is in NF and for any $s, h \models \Delta$, then $dom(h)$ is uniquely defined by s.

The normalisation rules are presented in Fig. 2. Basically, ω-ENT applies these rules to a leaf exhaustively and transforms it into NF before others. Given an inductive predicate $\mathtt{P}(E, F, ...)$, rule ExM excludes the middle by doing case analysis for the predicate between base-case (i.e., $E{=}F$) and recursive-case (i.e., $E{\neq}F$). The normalisation rule $\neq$null follows the following facts: $E{\mapsto}c(\_) \Rightarrow E{\neq}\mathtt{null}$ and $\mathtt{P}(E,F,\_){\wedge}E{\neq}F \Rightarrow E{\neq}\mathtt{null}$. Similarly, rule $\neq*$ follows the following facts: $x{\mapsto}\_{*}\mathtt{P}(y,F,\_){\wedge}y{\neq}F \Rightarrow x{\neq}y$, $x{\mapsto}\_{*}y{\mapsto}\_ \Rightarrow x{\neq}y$, and $\mathtt{P_i}(x,F_1,\_){*}\mathtt{P_j}(y,F_2,\_){\wedge}x{\neq}F_1{\wedge}y{\neq}F_2 \Rightarrow x{\neq}y$.

Axiom and Reduction Axiom rules include Emp, Inconsistency and Id, presented in Fig. 3. If each of these rules is applied to a leaf node, the node is evaluated as valid and marked as closed. The remaining ones in Fig. 3 are reduction rules.

For simplicity, the unfoldings in rules Frame, RInd, and LInd are applied with the following definition of inductive predicates:

$$\begin{aligned}&\mathtt{P}(x,F,\bar{B},u,sc,tg) \equiv \mathtt{emp}{\wedge}x{=}F{\wedge}sc{=}tg\\&\quad\vee\ \exists X,sc',d_1,d_2.x{\mapsto}c(X,d_1,d_2,u,sc){*}\mathtt{Q_1}(d_1,B){*}\mathtt{Q_2}(d_2,X){*}\mathtt{P}(X,F,\bar{B},u,sc',tg){\wedge}\pi_0\end{aligned}$$

where $B{\in}\bar{B}$, the matrix κ' contains two nested predicates Q_1 and Q_2, and the heap cell $c \in Node$ is defined as data $c\{c\ next; c_1\ down_1; c_2\ down_2; \tau_s\ scdata; \tau_u\ udata\}$ where $c_1, c_2{\in}Node$, $down_1$ and $down_2$ fields are for the nested predicates in the matrix

$$\mathtt{Subst}\ \frac{\Delta[E/x] \vdash \Delta'[E/x]}{\Delta{\wedge}x{=}E \vdash \Delta'} \qquad \text{ExM}\ \frac{\Delta{\wedge}E_1{=}E_2 \vdash \Delta' \quad \Delta{\wedge}E_1{\neq}E_2 \vdash \Delta'}{\Delta \vdash \Delta'}\ \begin{array}{c}E_1{=}E_2,\, E_1{\neq}E_2{\notin}\pi\ \&\\ FV(E_1, E_2) \subseteq (FV(\Delta){\cup}FV(\Delta'))^S\end{array}$$

$$=\!\text{L}\ \frac{\Delta \vdash \Delta'}{\Delta{\wedge}E{=}E \vdash \Delta'} \qquad \text{LBase}\ \frac{(\kappa{\wedge}\pi)[tg/sc] \vdash \Delta'[tg/sc]}{\mathtt{P}(E,E,\bar{B},u,sc,tg){*}\kappa{\wedge}\pi \vdash \Delta'}$$

$$\neq\!\mathtt{null}\ \frac{op(E){*}\kappa{\wedge}\pi{\wedge}G(op(E)){\wedge}E{\neq}\mathtt{null} \vdash \Delta'}{op(E){*}\kappa{\wedge}\pi{\wedge}G(op(E)) \vdash \Delta'}\ E{\neq}\mathtt{null}{\notin}\pi$$

$$\neq\!*\ \frac{op_1(E_1){*}op_2(E_2){*}\kappa{\wedge}\pi{\wedge}E_1{\neq}E_2 \vdash \Delta'}{op_1(E_1){*}op_2(E_2){*}\kappa{\wedge}\pi \vdash \Delta'}\ E_1{\neq}E_2{\notin}\pi \text{ and } G(op_1(E_1)), G(op_2(E_2)) \in \pi$$

Fig. 2: Normalization rules

$$\text{Id}\ \frac{}{\Delta \wedge \pi \vdash \Delta} \qquad \text{Emp}\ \frac{}{\texttt{emp}{\wedge}\pi \vdash \texttt{emp}{\wedge}\texttt{true}} \qquad \text{Inconsistency}\ \frac{}{\kappa{\wedge}\pi \vdash \Delta}\ \pi \models \texttt{false}$$

$$\text{=R}\ \frac{\Delta \vdash \Delta'}{\Delta \vdash \Delta'{\wedge}E{=}E} \qquad \text{Hypothesis}\ \frac{\Delta{\wedge}\pi \vdash \Delta'}{\Delta{\wedge}\pi \vdash \Delta'{\wedge}\pi'}\ \pi \models \pi' \qquad \text{RBase}\ \frac{\Delta \vdash \Delta' \wedge tg{=}sc}{\Delta \vdash \texttt{P}(E,E,\bar{B},u,sc,tg){*}\Delta'}$$

$$* \ \frac{\kappa_1{\wedge}\pi \vdash \kappa_2 \quad \kappa{\wedge}\pi \vdash \kappa'{\wedge}\pi'}{\kappa_1{*}\kappa{\wedge}\pi \vdash \kappa_2{*}\kappa' \wedge \pi'}\ \begin{array}{c}\texttt{roots}(\kappa_1) \cap \texttt{roots}(\kappa) = \emptyset \ \&\ \mathit{FV}(\kappa_2){\subseteq}\mathit{FV}(\kappa_1{\wedge}\pi){\cup}\{\texttt{null}\} \\ \&\ \mathit{FV}(\kappa'){\subseteq}\mathit{FV}(\kappa{\wedge}\pi){\cup}\{\texttt{null}\}\end{array}$$

$$\text{Frame}\ \frac{\begin{array}{l}\texttt{Q}_1(E_1,B)^{0}{*}\texttt{Q}_2(E_2,X)^{0}{*}\texttt{P}(X,F,\bar{B},u,sc',tg)^{k}{*}\Delta_1{\wedge}x{\neq}F_3{\wedge}\pi_0 \\ \quad \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2{\wedge}\pi_2\end{array}}{\texttt{P}(x,F,\bar{B},u,sc,tg)^{k}{*}\Delta_1{\wedge}x{\neq}F_3 \vdash x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_2{\wedge}\pi_2}\ x{\mapsto}c(\_){\notin}\kappa_2$$

$$\text{RInd}\ \frac{\begin{array}{l}x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_1{\wedge}\pi_1{\wedge}x{\neq}F \\ \quad \vdash x{\mapsto}c(X,E_1,E_2,u,sc'){*}\texttt{Q}_1(E_1,B){*}\texttt{Q}_2(E_2,X){*}\texttt{P}(X,F,\bar{B},u,sc',tg){*}\kappa_2{\wedge}\pi_2{\wedge}\pi_0\end{array}}{x{\mapsto}c(X,E_1,E_2,u,sc'){*}\kappa_1{\wedge}\pi_1{\wedge}x{\neq}F \vdash \texttt{P}(x,F,\bar{B},u,sc,tg){*}\kappa_2{\wedge}\pi_2}\ \dagger$$

$$\text{LInd}\ \frac{\begin{array}{l}x{\mapsto}c(X,E_1,E_2,u,sc'){*}\texttt{Q}_1(E_1,B)^{0}{*}\texttt{Q}_2(E_2,X)^{0}{*}\texttt{P}(X,F,\bar{B},u,sc',tg)^{k+1}{*}\Delta_1{\wedge}x{\neq}F_3{\wedge}\pi_0 \\ \quad \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2{\wedge}\pi_2\end{array}}{\texttt{P}(x,F,\bar{B},u,sc,tg)^{k}{*}\Delta_1{\wedge}x{\neq}F_3 \vdash \texttt{Q}(x,F_3,\bar{B},u,sc,tg_2){*}\kappa_2{\wedge}\pi_2}\ \sharp$$

Fig. 3: Reduction rules (where $\sharp$: $\texttt{P}(x,F,\bar{B},u,sc,tg){\notin}\kappa_2$, $\dagger$: $x{\mapsto}c(X,E_1,E_2,u,sc'){\notin}\kappa_2$)

heaps, the *udata* field is for the transitivity data, and the *scdata* field is for ordering data. The rules for the general form of the matrix heaps κ' are presented in [28].

=R and Hypothesis eliminate pure constraints in the RHS. In rule $*$, $\texttt{roots}(\kappa)$ is defined inductively as: $\texttt{roots}(\texttt{emp}){\equiv}\{\}$, $\texttt{roots}(r{\mapsto}\_){\equiv}\{r\}$, $\texttt{roots}(P(r,F,..)){\equiv}\{r\}$ and $\texttt{roots}(\kappa_1{*}\kappa_2) \equiv \texttt{roots}(\kappa_1){\cup}\texttt{roots}(\kappa_2)$. This rule is applied in three ways. First, it is applied into an entailment which is of the form $\kappa{\wedge}\pi \vdash \kappa{\wedge}\pi'$. It matches and discards the identified heap predicates between the two sides to generate a premise with empty heaps. As a result, this premise may be applied with the axiom rule EMP. Secondly, it is applied to an entailment of the form $x_i{\mapsto}c_i(\bar{v}_i){*}...{*}x_n{\mapsto}c_n(\bar{v}_n){\wedge}\pi \vdash \kappa'{\wedge}\pi'$. For each points-to predicate $x_i{\mapsto}c_i(\bar{v}_i){\in}\kappa'$, ω-ENT searches for one points-to predicate $x_j{\mapsto}c_j(\bar{v}_j)$ in the LHS such that $x_j{\mapsto}c_j(\bar{v}_j) \equiv x_i{\mapsto}c_i(\bar{v}_i)$. Lastly, it is applied into an entailment that is of the form $\Delta_1 * \Delta \vdash \Delta_2 * \Delta'$ where either $\Delta_1 \vdash \Delta_2$ or $\Delta \vdash \Delta'$ could be linked back into an internal node.

In RInd, for each occurrence of inductive predicates $\texttt{P}(r,F,\bar{B},u,sc,tg)$ in κ', ω-ENT searches for a points-to predicate $r{\mapsto}\_$. If any of these searches fail, ω-ENT decides the conclusion as invalid. Rule LInd unfolds the inductive predicates in the LHS. Every LHS of entailments in this rule also captures the unfolding numbers for the subterm relationship and generates the progressing point in the cyclic proofs afterwards. These numbers are essential for our system to construct cyclic proofs. This rule is applied in a *depth-first* manner, i.e., if there are more than one occurrences of inductive predicates in the LHS that could be applied by this rule, the one with the greatest unfolding number is chosen. We emphasise that the last five rules still work well when the predicate in the RHS contains only a subset of the local properties wrt. the predicate in the LHS.

Back-Link Generation Procedure $\texttt{link\_back}_{\texttt{e}}$ generates a back-link as follows. In a preproof, given a path containing a back-link, say $\mathtt{e}_1, \mathtt{e}_2, .., \mathtt{e}_m$ where $\mathtt{e}_1$ is a companion and $\mathtt{e}_m$ a bud, then $\mathtt{e}_1$ is in NF and of the following form:

- $\mathtt{e}_1{\equiv}\mathtt{P}(x{,}F{,}\bar{B}{,}u{,}sc{,}tg)^k{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null} \vdash \mathtt{Q}(x{,}F_2{,}\bar{B}{,}u{,}sc{,}tg_2){*}\kappa'{\wedge}\pi'$.
- $\mathtt{e}_2$ is obtained from applying $\mathtt{LInd}$ into $\mathtt{e}_1$. $\mathtt{e}_2$ is of the form:

$$x{\mapsto}c(X, \bar{p}, {,}u{,}sc){*}\kappa'{*}\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1 \\ \vdash \mathtt{Q}(x{,}F_2{,}\bar{B}{,}u{,}sc{,}tg_2){*}\kappa'{\wedge}\pi'$$

  We remark that $sc \diamond sc' \in \pi_1$, and if $k \geq 1$, then $sc_i \diamond sc \in \pi$
- $\mathtt{e}_3$, .., $\mathtt{e}_{m-4}$ are obtained from applications of normalisation rules to normalise the LHS of $\mathtt{e}_2$ due to the presence of κ'. As the roots of inductive predicates in κ' are fresh variables, the applications of the normalization rules above do not affect the RHS of $\mathtt{e}_2$. That means the RHS of $\mathtt{e}_3$, .., and $\mathtt{e}_{m-4}$ are the same as that of $\mathtt{e}_2$. As a result, $\mathtt{e}_{m-4}$ is of the form:

$$x{\mapsto}c(X, \bar{p}, {,}u{,}sc){*}\kappa''_1{*}\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1{\wedge}\pi_2 \\ \vdash \mathtt{Q}(x{,}F_2{,}\bar{B}{,}u{,}sc{,}tg_2){*}\kappa'{\wedge}\pi'$$

  where κ''_1 may be $\mathtt{emp}$ and π_2 is a conjunction of disequalities coming from $\mathtt{ExM}$.
- $\mathtt{e}_{m-3}$ is obtained from the application of $\mathtt{ExM}$ over x and F_2 and of the form:

$$x{\mapsto}c(X, \bar{p}, {,}u{,}sc){*}\kappa''_1{*}\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1{\wedge}\pi_2 \\ {\wedge}x{\neq}F_2 \vdash \mathtt{Q}(x{,}F_2{,}\bar{B}{,}u{,}sc{,}tg_2){*}\kappa'{\wedge}\pi'$$

  (For the case $x{=}F_2$, the rule $\mathtt{ExM}$ is kept applying until either $F \equiv F_2$, that is, two sides are reaching the end of the same heap segment, or it is stuck.)
- $\mathtt{e}_{m-2}$ is obtained from the application of $\mathtt{RInd}$ and is of the form:

$$x{\mapsto}c(X, \bar{p}, {,}u{,}sc){*}\kappa''_1{*}\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1{\wedge}\pi_2 \\ {\wedge}x{\neq}F_2 \vdash x{\mapsto}c(X{,}\bar{p}{,}u{,}sc){*}\kappa''_2{*}\mathtt{Q}(X{,}F_2{,}\bar{B}{,}u{,}sc'{,}tg_2){*}\kappa'{\wedge}\pi'{\wedge}\pi'_2$$

- $\mathtt{e}_{m-1}$ is obtained from the application of the $\mathtt{Hypothesis}$ to eliminate π'_2 (otherwise, it is stuck) and is of the form:

$$x{\mapsto}c(X, \bar{p}, {,}u{,}sc){*}\kappa''_1{*}\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1{\wedge}\pi_2 \\ {\wedge}x{\neq}F_2 \vdash x{\mapsto}c(X{,}\bar{p}{,}u{,}sc){*}\kappa''_2{*}\mathtt{Q}(X{,}F_2{,}\bar{B}{,}u{,}sc'{,}tg_2){*}\kappa'{\wedge}\pi'$$

- $\mathtt{e}_m$ is obtained from the application of $*$ and is of the form:

$$\mathtt{P}(X{,}F{,}\bar{B}{,}u{,}sc'{,}tg)^{k+1}{*}\kappa{\wedge}\pi{\wedge}x{\neq}F{\wedge}x{\neq}\mathtt{null}{\wedge}\pi_1{\wedge}\pi_2{\wedge}x{\neq}F_2 \\ \vdash \mathtt{Q}(X{,}F_2{,}\bar{B}{,}u{,}sc'{,}tg_2){*}\kappa'{\wedge}\pi'$$

When $k \geq 1$, it is always possible to link $\mathtt{e}_m$ back to $\mathtt{e}_1$ through the substitution is $\sigma{\equiv}[x/X, sc/sc']$ after weakening some pure constraints in its LHS.

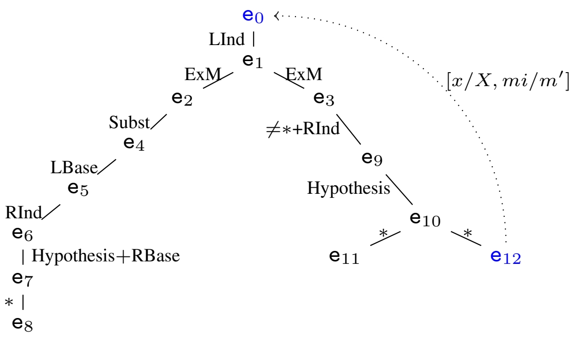

Fig. 4: Cyclic Proof of $\texttt{lls}(x,\texttt{null},mi,ma)^0 \wedge x{\neq}\texttt{null} \vdash \texttt{llb}(x,\texttt{null},mi)$.

4.2 Illustrative Example

We illustrate our system through the following example:

$$\texttt{e}_0\text{: } \texttt{lls}(x,\texttt{null},mi,ma)^0 \wedge x{\neq}\texttt{null} \vdash \texttt{llb}(x,\texttt{null},mi)$$

where the sorted linked-list lls (mi is the minimum value and ma is the maximum value) is defined in Sect. 2.1 and llb define singly-linked lists whose values are greater than or equal to a constant number. Particularly, predicate llb is defined as follows.

$$\begin{aligned}&\texttt{pred llb}(r,F,b) \equiv \texttt{emp}{\wedge}r{=}F\\ &\quad \vee \exists X_{tl},d.r{\mapsto}c_4(X_{tl},d) * \texttt{llb}(X_{tl},F,b){\wedge}r{\neq}F \wedge b{\leq}d\end{aligned}$$

Since the LHS is stronger than the RHS, this entailment is valid. Our system could generate the cyclic proof (shown in Fig. 4) to prove the validity of $\texttt{e}_0$. In the following, we present step-by-step to show how the proof was created. Firstly, $\texttt{e}_0$, which is in NF, is applied with rule LInd to unfold predicate $\texttt{lls}(x,\texttt{null},mi,ma)^0$ and obtain $\texttt{e}_1$ as:

$$\texttt{e}_1\text{: } x{\mapsto}c_4(X,m') * \texttt{lls}(X,\texttt{null},m',ma)^1 \wedge x{\neq}\texttt{null} \wedge mi{\leq}m' \vdash \texttt{llb}(x,\texttt{null},mi)$$

We remark that the unfolding number of the recursive predicate lls in the LHS is increased by 1. Next, our system normalizes $\texttt{e}_1$ by applying rule ExM into X and null to generate two children, $\texttt{e}_2$ and $\texttt{e}_3$, as follows.

$$\begin{aligned}&\texttt{e}_2\text{: } x{\mapsto}c_4(X,m') * \texttt{lls}(X,\texttt{null},m',ma)^1 \wedge x{\neq}\texttt{null} \wedge mi{\leq}m' \wedge X{=}\texttt{null}\\ &\qquad \vdash \texttt{llb}(x,\texttt{null},mi)\\ &\texttt{e}_3\text{: } x{\mapsto}c_4(X,m') * \texttt{lla}(X,\texttt{null},m',ma)^1 \wedge x{\neq}\texttt{null} \wedge mi{\leq}m' \wedge X{\neq}\texttt{null}\\ &\qquad \vdash \texttt{llb}(x,\texttt{null},mi)\end{aligned}$$

For the left child, it applies normalization rules to obtain $\texttt{e}_4$ (substitute X by null) and then $\texttt{e}_5$, by LBase to unfold $\texttt{lls}(\texttt{null},\texttt{null},m',ma)^1$ to the base case, as:

$$\begin{aligned}&\texttt{e}_4\text{: } x{\mapsto}c_4(\texttt{null},m') * \texttt{lls}(\texttt{null},\texttt{null},m',ma)^1 \wedge x{\neq}\texttt{null} \wedge mi{\leq}m' \vdash \texttt{llb}(x,\texttt{null},mi)\\ &\texttt{e}_5\text{: } x{\mapsto}c_4(\texttt{null},ma) \wedge x{\neq}\texttt{null} \wedge mi{\leq}ma \vdash \texttt{llb}(x,\texttt{null},mi)\end{aligned}$$

Now, $\mathtt{e}_5$ is in NF. $\mathtt{S2S}_{\mathtt{Lin}}$ applies `RInd` and then `RBase` to `llb` in the RHS as:

$$\begin{aligned}\mathtt{e}_6\text{: } & x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \\ & \vdash x{\mapsto}c_4(\mathtt{null}, ma) * \mathtt{llb}(\mathtt{null},\mathtt{null},mi) \wedge mi{\leq}ma \\ \mathtt{e}_{6'}\text{: } & x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash x{\mapsto}c_4(\mathtt{null}, ma){\wedge}mi{\leq}ma\end{aligned}$$

After that, as $mi{\leq}ma \Rightarrow mi{\leq}ma$, $\mathtt{e}_{6'}$ is applied with `Hypothesis` to obtain $\mathtt{e}_7$.

$$\mathtt{e}_7\text{: } x{\mapsto}c_4(\mathtt{null}, ma) \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash x{\mapsto}c_4(\mathtt{null}, ma)$$

As the LHS of $\mathtt{e}_7$ is in NF and a base formula, it is sound and complete to apply rule $*$ to have $\mathtt{e}_8$ as $\mathtt{emp} \wedge x{\neq}\mathtt{null} \wedge mi{\leq}ma \vdash \mathtt{emp}$. By `Emp`, $\mathtt{e}_8$ is decided as `valid`. For the right branch of the proof, $\mathtt{e}_3$ is applied with rule $\neq*$ and then `RInd` to obtain $\mathtt{e}_9$:

$$\begin{aligned}\mathtt{e}_9\text{: } & x{\mapsto}c_4(X, m'){*}\mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X \\ & \vdash x{\mapsto}c_4(X, m'){*}\mathtt{llb}(X,\mathtt{null},mi){\wedge}mi{\leq}m'\end{aligned}$$

Then, $\mathtt{e}_9$ is applied with `Hypothesis` to eliminate the pure constraint in the RHS:

$$\begin{aligned}\mathtt{e}_{10}\text{: } & x{\mapsto}c_4(X, m'){*}\mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X \\ & \vdash x{\mapsto}c_4(X, m'){*}\mathtt{llb}(X,\mathtt{null},mi)\end{aligned}$$

$\mathtt{e}_{10}$ is then applied the rule $*$ to obtain $\mathtt{e}_{11}$ and $\mathtt{e}_{12}$ as follows.

$$\begin{aligned}&\mathtt{e}_{11}\text{: } x{\mapsto}c_4(X, m') \vdash x{\mapsto}c_4(X, m') \\ &\mathtt{e}_{12}\text{: } \mathtt{lls}(X,\mathtt{null},m',ma)^1 \wedge x{\neq}\mathtt{null} \wedge mi{\leq}m' \wedge X{\neq}\mathtt{null} \wedge x{\neq}X \vdash \mathtt{llb}(X,\mathtt{null},mi)\end{aligned}$$

$\mathtt{e}_{11}$ is valid by `Id`. $\mathtt{e}_{12}$ is successfully linked back to $\mathtt{e}_0$ to form a pre-proof as

$$(\mathtt{lls}(X,\mathtt{null},m',ma)^1{\wedge}X{\neq}\mathtt{null})[x/X, mi/m'] \vdash \mathtt{llb}(X,\mathtt{null},mi)[x/X, mi/m']$$

is identical to $\mathtt{e}_0$. Since $\mathtt{lls}(X,\mathtt{null},m',ma)^1$ in $\mathtt{e}_{12}$ is the subterm of $\mathtt{lls}(x,\mathtt{null},mi,ma)^0$ in $\mathtt{e}_0$, our system decided that $\mathtt{e}_0$ is valid with the cyclic proof presented in Fig. 4.

5 Soundness, Completeness, and Complexity

We describe the soundness, termination, and completeness of ω-ENT. First, we need to show the invariant about the quantifier-free entailments of our system.

Corollary 1. *Every entailment derived from ω-ENT is quantifier-free.*

The following lemma shows the soundness of the proof rules.

Lemma 1 (Soundness). *For each proof rule, the conclusion is valid if all premises are valid.*

As every backlink generated contains at least one pair of inductive predicate occurrences in a subterm relationship, the global soundness condition holds in our system.

Lemma 2 (Global Soundness). *A pre-proof derived is indeed a cyclic proof.*

The termination relies on the number of premises/entailments generated by $*$. As the number of inductive symbols and their arities are finite, there is a finite number of equivalence classes of these entailments in which any two entailments in the same class are equivalent under some substitution and linked back together. Therefore, the number of premises generated by the rule $*$ is finite, considering the back-links generation.

Lemma 3. ω-$\mathtt{ENT}$ *terminates.*

In the following, we show the complexity analysis. First, we show that every occurrence of inductive predicates in the LHS is unfolded at most two times.

Lemma 4. *Given any entailment* $\mathtt{P}(\bar{v})^k * \Delta_a \vdash \Delta_c$, $0 \leq k \leq 2$.

Let n be the maximum number of predicates (both inductive predicates and points-to predicates) among the LHS of the input and the definitions in $\mathcal{P}$, and m be the maximum number of fields of data structures. Then, the complexity is defined as follows.

Proposition 1 (Complexity). $\mathtt{QF\_ENT{-}SL_{LIN}}$ *is* $\mathcal{O}(n \times 2^m + n^3)$.

If m is bounded by a constant, the complexity becomes polynomial in time.

Our completeness proofs are shown in two steps. First, we show the proofs for an entailment whose LHS is a base formula. Second, we show the correctness when the LHS contains inductive predicates. In the following, we first define the base formulas of the LHS derived by ω-$\mathtt{ENT}$ from occurrences of inductive predicates. Based on that, we define bad models to capture counter-models of invalid entailments.

Definition 4 (`SHLIDe` **Base)** *Given* κ, *define* $\overline{\kappa}$ *as follows.*

$$\overline{\mathtt{P}(E,F,\bar{B},u,sc,tg)} \overset{def}{=} E{\mapsto}c(F,E_1,E_2,u,tg) * \overline{\mathtt{Q}_1(E_1,B)}*\overline{\mathtt{Q}_2(E_2,F)}\wedge\pi_0$$
$$\overline{E{\mapsto}c(\bar{v})} \overset{def}{=} E{\mapsto}c(\bar{v}) \qquad \overline{\mathtt{emp}} \overset{def}{=} \mathtt{emp} \qquad \overline{\kappa_1{*}\kappa_2} \overset{def}{=} \overline{\kappa_1}{*}\overline{\kappa_2}$$

The definition for general predicates with arbitrary matrix heaps is presented in [28]. As $\mathcal{P}$ does not include mutual recursion (Condition **C3**), the definition above terminates in a finite number of steps. In a pre-proof, these `SHLIDe` base formulas of the LHS are obtained once every inductive predicate has been unfolded.

Lemma 5. *If* $\kappa \wedge \pi$ *is in NF, then* $\overline{\kappa} \wedge \pi$ *is in NF, and* $\overline{\kappa} \wedge \pi \vdash \kappa$ *is valid.*

In other words, $\overline{\kappa} \wedge \pi$ is an under-approximation of $\kappa \wedge \pi$; invalidity of $\overline{\kappa} \wedge \pi \vdash \Delta'$ implies invalidity of $\kappa \wedge \pi \vdash \Delta'$.

Definition 5 (Bad Model) *The bad model for* $\overline{\kappa} \wedge \phi \wedge a$ *in NF is obtained by assigning*

- *a distinct non-*`null` *value to each variable in FV*$(\overline{\kappa} \wedge \phi)$*; and*
- *a value to each variable in FV*(a) *such that* a *is satisfiable.*

Lemma 6. 1. *For every proof rule except the rule* $*$*, all premises are valid only if the conclusion is valid.*
2. *For the rule* $*$*, where the conclusion is of the form* $\Delta^b \vdash \kappa'$*, all premises are valid only if the conclusion is valid and* Δ^b *is in NF.*

The following lemma states that the correctness of the procedure `is_closed` for cases 2(b-d).

Lemma 7 (Stuck Invalidity). *Given $\kappa \wedge \pi \vdash \Delta'$ in NF, it is* `invalid` *if the procedure* `is_closed` *returns* `invalid` *for cases 2(b-d).*

A bad model of the $\overline{\kappa} \wedge \pi$ is a counter-model. Cases 2b) and 2c) show that the heaps of bad models are not connected, and thus accordingly to conditions **C1** and **C2**, any model of the LHS could not be a model of the RHS. Case 2d) shows that heaps of the two sides could not be matched. We next show the correctness of Case 2(a) of the procedure `is_closed`, and invalidity is preserved during the proof search in ω-ENT.

Proposition 2 (Invalidity Preservation). *If ω-ENT is stuck, the input is invalid.*

In other words, if ω-ENT returns invalid, we can construct a bad model.

Theorem 2. $\mathtt{QF\_ENT-SL_{LIN}}$ *is decidable.*

6 Implementation and Evaluation

We implement $\mathtt{S2S_{Lin}}$ using OCaml. This implementation is an instantiation of a general framework for cyclic proofs. We utilize the cyclic proof systems to derive bases for inductive predicates shown in [24] to discharge satisfiability of separation logic formulas. We use the solver presented in [29,31] for those formulas beyond this fragment. We also develop a built-in solver for discharging equalities.

We evaluated $\mathtt{S2S_{Lin}}$ to show that i) it can discharge problems in `SHLIDe` effectively; and ii) its performance is compatible with state-of-the-art solvers. The evaluation of $\mathtt{S2S_{Lin}}$ is provided as a companion artifact [27].

Experiment settings We have evaluated $\mathtt{S2S_{Lin}}$ on entailment problems taken from SL-COMP benchmarks [38], a competition of separation logic solvers. We take 356 problems (out of 983) in two divisions of the competition, *qf_shls_entl* and *qf_shlid_entl*, and one new division, *qf_shlid2_entl*. All these problems semantically belong to our decidable fragment, and their syntax is written in SMT 2.6 format [39].

- Division *qf_shls_entl* includes 296 entailment problems, 122 `invalid` problems and 174 `valid` problems, with only singly-linked lists. The authors in [33] randomly generated them
- Division *qf_shlid_entl* contains 60 entailment problems which the authors in [15] handcrafted. They include singly-linked lists, doubly-linked lists, lists of singly-linked lists, or skip lists. Furthermore, the system of inductive predicates must satisfy the following condition: For two different predicates `P`, `Q` in the system of definitions, either $\mathtt{P} \prec^*_{\mathcal{P}} \mathtt{Q}$ or $\mathtt{Q} \prec^*_{\mathcal{P}} \mathtt{P}$.
- In the third division, we introduce new benchmarks, with 27 problems, beyond the above two divisions. In particular, every system of predicate definitions includes two predicates, `P` and `Q`, that are semantically equivalent. We have submitted this division to the Github repository of SL-COMP.

Table 1: Experimental results

Tool	*qf_shls_entl*			*qf_shlid_entl*			*qf_shlid2_entl*		
	`invalid` (122)	`valid` (174)	Time (296)	`invalid` (24)	`valid` (36)	Time (60)	`invalid` (14)	`valid` (13)	Time (27)
SLS	12	174	507m42s	2	35	133m28s	0	11	97m54s
`Spen`	122	174	10.78s	14	13	3.44s	8	2	1.69s
`Cyclist`$_{\texttt{SL}}$	0	58	1520m5s	0	24	360m38s	0	3	240m3s
Harrsh	39	116	425m19s	18	27	53m56s	8	7	156m45s
`Songbird`	12	174	237m25s	2	35	40m38s	0	12	47m11s
`S2S`$_{\texttt{Lin}}$	122	174	6.22s	24	36	0.96s	14	13	1.20s

To evaluate `S2S`$_{\texttt{Lin}}$'s performance, we compared it with the state-of-the-art tools such as `Cyclist`$_{\texttt{SL}}$ [5], `Spen` [15], `Songbird` [40], SLS [41] and Harrsh [23]. We omitted Cycomp [42], as these benchmarks are beyond its decidable fragment. Note that `Cyclist`$_{\texttt{SL}}$, `Songbird` and SLS are not complete; for non-valid problems, while `Cyclist`$_{\texttt{SL}}$ returns `unknown`, `Songbird` and SLS use some heuristic to guess the outcome. For each division, we report the number of correct outputs (`invalid`, `valid`) and the time (in minutes and seconds) taken by each tool. Note that we use the status (`invalid`, `valid`) annotated with each problem in the SL-COMP benchmark as the ground truth. If the output is the same as the status, we classify it as correct; otherwise, it is marked as incorrect. We also note that in these experiments, we used the competition pre-processing tool [39] to transform the SMT 2.6 format into the corresponding formats of the tools before running them. All experiments were performed on an Intel Core i7-6700 CPU 3.4Gh and 8GB RAM. The CPU timeout is 600 seconds.

Experiment results The experimental results are reported in Table 1. In this table, the first column presents the names of the tools. The following three columns show the results of the first division, including the number of correct `invalid` outputs, the number of correct `valid` outputs and the taken time (where *m* for minutes and *s* for seconds), respectively. The number between each pair of brackets *(...)* in the third row shows the number of problems in the corresponding column. Similarly, the following two groups of six columns describe the results of the second and third divisions, respectively.

In general, the experimental results show that `S2S`$_{\texttt{Lin}}$ is the one (and only one) that could produce all the correct results. Other solvers either produced wrong results or could discharge a fraction of the experiments. Moreover, `S2S`$_{\texttt{Lin}}$ took a short time for the experiments (8.38 seconds compared to 15.91 seconds for `Spen`, 324 minutes for `Songbird`, 635 minutes for Harrsh, 739 minutes for SLS and 2120 minutes for `Cyclist`$_{\texttt{SL}}$). While SLS returned 14 false negatives, `Spen` reported 20 false positives. `Cyclist`$_{\texttt{SL}}$, `Songbird` and Harrsh did not produce any wrong results. Of 569 tests, `Cyclist`$_{\texttt{SL}}$ could handle 85 tests (15%), Harrsh could handle 215 tests (38%), and `Songbird` could decide on 235 tests (41.3%). In the total of 223 `valid` tests, `Cyclist`$_{\texttt{SL}}$ could handle 85 problems (38%), and `Songbird` could decide 222 problems (99.5%).

Now we examine the results for each division in detail. For *qf_shls_entl*, `Spen` returned all correct, `Songbird` 186, Harrsh 155, and `Cyclist`$_{\texttt{SL}}$ 58. If we set the timeout to 2400 seconds, both `Songbird` and Harrsh produced all the correct results. Division *qf_shlid_entl* includes 24 `invalid` problems and 36 `valid` problems. While `Songbird` produced 37 problems correctly, `Cyclist`$_{\texttt{SL}}$ produced 24 correct results. `Spen` reported 27 correct results and 13 false positives (`skl2−vc`$\{$`01` $-$ `04`$\}$ `skl3−vc01`, `skl3−vc`$\{$`03` $-$ `10`$\}$). The last division, *qf_shlid2_entl*, includes 14 `invalid` and 13 `valid` test problems. While `Songbird` decided only 12 problems correctly, `Cyclist`$_{\texttt{SL}}$ produced 3 correct outcomes. `Spen` reported 10 correct results. However, it produced 7 false positives (`ls−mul−vc`$\{$`01` $-$ `03`$\}$, `ls−mul−vc05`, `nll−mul−vc`$\{$`01` $-$ `03`$\}$). We believe that engineering design and effort play an essential role alongside theory development. Since our experiments provide breakdown results of the two SL-COMP competition divisions, we hope that they provide an initial understanding of the SL-COMP benchmarks and tools. Consequently, this might reduce the effort to prepare experiments over these benchmarks to evaluate new SL solvers. Finally, one might point out that `S2S`$_{\texttt{Lin}}$ performed well because the entailments in the experiments are within its scope. We do not entirely disagree with this argument but would like to emphasize that tools do not always work well on favourable benchmarks. For example, `Spen` introduced wrong results on *qf_shlid_entl*, and Harrsh did not handle *qf_shlid_entl* and *qf_shlid2_entl* well, although these problems are in their decidable fragments.

7 Related Work

`S2S`$_{\texttt{Lin}}$ is a variant of the cyclic proof systems [3,4,5,26] and [42]. Unlike existing cyclic proof systems, the soundness of `S2S`$_{\texttt{Lin}}$ is local, and the proof search is not backtracking. The work presented in [42] shows the completeness of the cyclic proof system. Its main contribution is introducing the rule $*$ for those entailments with a disjunction in the RHS obtained from predicate unfolding. In contrast to [42], our work includes normalization to soundly and completely avoid disjunction in the RHS during unfolding. Moreover, our decidable fragment SHLIDe is non-overlapping to the cone predicates introduced in [42]. Furthermore, due to the empty heap in the base cases, the matching rule in [42] cannot be applied to the predicates in `SHLIDe`. Finally, our work also presents how to obtain the global soundness condition for cyclic proofs.

Our work relates to the inductive theorem provers introduced in [10], [40] and Smallfoot [2]. While [10] is based on structural induction, [40] is based on mathematical induction. Smallfoot [2] proposed a decision procedure for linked lists and trees. It used a fixed compositional rule as a consequence of induction reasoning to handle inductive entailments. Compared with Smallfoot, our proof system replaces the compositional rule by combining rule `LInd` and the back-link construction. Our system could support induction reasoning on a much more expressive fragment of inductive predicates.

Our proposal also relates to works that use lemmas as consequences of induction reasoning [2,16,30,41]. These works in [16,25,30,41] automatically generate lemmas for some classes of inductive predicates. S2 [25] generated lemmas to normalize (such as split and equivalence) the shapes of the synthesized data structures. [16] proposed to generate several sets of lemmas not only for compositional predicates but also for different predicates (e.g., completion lemmas, stronger lemmas and static parameter contraction lemmas). SLS [41] aims to infer general lemmas to prove an entailment. Similarly, S2ENT [30] solves a more generic problem, frame inference, using cyclic proofs and lemma synthesis. It infers a shape-based residual frame in the LHS and then synthesizes the pure constraints over the two sides.

$\texttt{S2S}_{\texttt{Lin}}$ relates to model-based decision procedures that reduce the entailment problem in separation logic to a well-studied problem in other domains. For instance, in [8,11,17], the entailment problem, including singly-linked lists and their invariants, is reduced to the problem of inclusion checking in a graph theory. The authors in [18] reduced the entailment problem to the satisfiability problem in second-order monadic logic. This reduction could handle an expressive fragment of spatial-based predicates called bounded-tree width. Moreover, the work presented in [23] shows a model-based decision procedure for a subfragment of the bounded-tree width. Furthermore, while the work in [15,19] reduced the entailment problem to the inclusion checking problem in tree automata, [21] presented an idea to reduce the problem to the inclusion checking problem in heap automata. Moreover, while the procedure in [15] supported compositional predicates (single and double links) well, the procedure in [19] could handle predicates satisfying local properties (e.g., trees with parent pointers). Our decidable fragment subsumes the one described in [2,11,15] but is incomparable to the ones presented in [8,17,18,19]. Works in [34] and [35,36] reduced the entailment problem in separation logic into the satisfiability problem in SMT. While GRASShoper [35,36] could handle transitive closure pure properties, $\texttt{S2S}_{\texttt{Lin}}$ is capable of supporting local ones. Unlike GRASShoper, which reduces entailment into SMT problems, $\texttt{S2S}_{\texttt{Lin}}$ reduces an entailment to admissible entailments and detects repetitions via cyclic proofs.

Decidable fragments and complexity results of the entailment problem in separation logic with inductive predicates were well studied. The entailment is 2-EXPTIME in cone predicates [42], the bounded tree-width predicates and beyond [18,14], and EXPTIME in a sub-fragment of cone predicates [19]. In the other class, entailment is in polynomial time for singly-linked lists [11] and semantically linear inductive predicates [15]. Moreover, the extensions with arithmetic [17] are in polynomial but become EXPTIME when the lists are extended with double links [8]. SHLIDe (with nested lists, trees and arithmetic properties) is roughly in the "middle" of the two classes above. The entailment is EXPTIME and becomes polynomial under the upper bound restriction.

8 Conclusion

We have presented a novel decision procedure for the quantifier-free entailment problem in separation logic combined with inductive definitions of compositional predicates and pure properties. Our proposal is the first complete cyclic proof system for the problem in separation logic without back-tracking. We have implemented the proposal in $\texttt{S2S}_{\texttt{Lin}}$ and evaluated it over the set of nontrivial entailments taken from the SL-COMP competition. The experimental results show that our proposal is effective and efficient when compared to the state-of-the-art solvers. For future work, we plan to develop a bi-abductive procedure based on an extension of this work with the cyclic frame inference procedure presented in [30]. This extension is fundamental to obtaining a compositional shape analysis beyond the lists and trees. Another work is to formally prove that our system is as strong as Smallfoot in the decidable fragment with lists and trees [2]: Given an entailment, if Smallfoot can produce proof, so is $\texttt{S2S}_{\texttt{Lin}}$.

References

1. Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures*, pages 411–425, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
2. J. Berdine, C. Calcagno, and P. W. O'Hearn. Symbolic Execution with Separation Logic. In *APLAS*, volume 3780, pages 52–68, November 2005.
3. J. Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *Proceedings of TABLEAUX-14*, volume 3702 of *LNAI*, pages 78–92. Springer-Verlag, 2005.
4. J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In *Proceedings of APLAS-10*, LNCS, pages 350–367. Springer, 2012.
5. James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *Proceedings of the 23rd International Conference on Automated Deduction*, CADE'11, page 131–146, Berlin, Heidelberg, 2011. Springer-Verlag.
6. Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, pages 3–11, Cham, 2015. Springer International Publishing.
7. Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. In *POPL*, pages 289–300, 2009.
8. Taolue Chen, Fu Song, and Zhilin Wu. Tractability of Separation Logic with Inductive Definitions: Beyond Lists. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
9. W.-N. Chin, C. Gherghina, R. Voicu, Q.-L. Le, F. Craciun, and S. Qin. A specialization calculus for pruning disjunctive predicates to support verification. In *CAV*. 2011.
10. Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. Automatic induction proofs of data-structures in imperative programs. In *Proceedings of PLDI*, PLDI '15, pages 457–466, New York, NY, USA, 2015. ACM.
11. B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR*, volume 6901, pages 235–249. 2011.
12. Christopher Curry, Quang Loc Le, and Shengchao Qin. Bi-abductive inference for shape and ordering properties. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 220–225, 2019.
13. Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. Scaling static analyses at facebook. *Commun. ACM*, 62(8):62–70, jul 2019.
14. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Unifying decidable entailments in separation logic with inductive definitions. In *Automated Deduction-CADE 28-28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, pages 183–199, 2021.
15. Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. *Formal Methods in System Design*, 51(3):575–607, 2017.
16. Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. On automated lemma generation for separation logic with inductive definitions. *ATVA*, 2015.
17. Xincai Gu, Taolue Chen, and Zhilin Wu. *A Complete Decision Procedure for Linearly Compositional Separation Logic with Data Constraints*, pages 532–549. Springer International Publishing, Cham, 2016.

18. R. Iosif, A. Rogalewicz, and J. Simácek. The tree width of separation logic with recursive definitions. In *CADE*, pages 21–38, 2013.
19. Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. *ATVA*, 2014.
20. S. Ishtiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *ACM POPL*, pages 14–26, London, January 2001.
21. Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. *Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic*, pages 611–638. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.
22. Katelaan Jens, Jovanovic Dejan, and Weissenbacher Georg. A separation logic with data: Small models and automation. In *IJCAI*, 2018.
23. Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 319–336, Cham, 2019. Springer International Publishing.
24. Quang Loc Le. Compositional satisfiability solving in separation logic. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 578–602, Cham, 2021. Springer International Publishing.
25. Quang Loc Le, Cristian Gherghina, Shengchao Qin, and Wei-Ngan Chin. Shape analysis via second-order bi-abduction. In *CAV*, volume 8559, pages 52–68. 2014.
26. Quang Loc Le and Mengda He. A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In Sukyoung Ryu, editor, *Programming Languages and Systems*, pages 350–372, Cham, 2018. Springer International Publishing.
27. Quang Loc Le and Xuan-Bach D. Le. Artifact for an efficient cyclic entailment procedure in a fragment of separation logic, February 2023. https://doi.org/10.5281/zenodo.7619870.
28. Quang Loc Le and Xuan-Bach D. Le. An efficient cyclic entailment procedure in a fragment of separation logic, January 2023. Technical Report.
29. Quang Loc Le, Jun Sun, and Wei-Ngan Chin. Satisfiability modulo heap-based programs. In *CAV*. 2016.
30. Quang Loc Le, Jun Sun, and Shengchao Qin. Frame inference for inductive entailment proofs in separation logic. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 41–60, 2018.
31. Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In *CAV*, pages 495–517, 2017.
32. Scott McPeak and George C. Necula. Data structure specifications via local equality axioms. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 476–490, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
33. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, page 556–566, New York, NY, USA, 2011. Association for Computing Machinery.
34. JuanAntonio Navarro Pérez and Andrey Rybalchenko. Separation logic modulo theories. In *APLAS*, volume 8301, pages 90–106. 2013.
35. R. Piskac, T. Wies, and D. Zufferey. Automating separation logic using smt. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044, pages 773–789. 2013.
36. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In *CAV*, volume 8559, pages 711–728. 2014.
37. J. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *IEEE LICS*, pages 55–74, 2002.

38. Mihaela Sighireanu and Quang Loc Le. SL-COMP 2022. https://sl-comp.github.io/, 2022. [Online; accessed Jun-2022].
39. Mihaela Sighireanu, Juan Antonio Navarro Pérez, Andrey Rybalchenko, Nikos Gorogiannis, Radu Iosif, Andrew Reynolds, Cristina Serban, Jens Katelaan, Christoph Matheja, Thomas Noll, Florian Zuleger, Wei-Ngan Chin, Quang Loc Le, Quang-Trung Ta, Ton-Chanh Le, Thanh-Toan Nguyen, Siau-Cheng Khoo, Michal Cyprian, Adam Rogalewicz, Tomás Vojnar, Constantin Enea, Ondrej Lengál, Chong Gao, and Zhilin Wu. SL-COMP: competition of solvers for separation logic. In *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics*, pages 116–132, 2019.
40. Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated mutual explicit induction proof in separation logic. In John Fitzgerald, Constance Heitmeyer, Stefania Gnesi, and Anna Philippou, editors, *FM 2016: Proceedings*, pages 659–676, 2016.
41. Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated lemma synthesis in symbolic-heap separation logic. *POPL*, 2018.
42. Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. Completeness of cyclic proofs for symbolic heaps with inductive definitions. In Anthony Widjaja Lin, editor, *Programming Languages and Systems*, pages 367–387, Cham, 2019. Springer International Publishing.