Separating the Wheat from the Chaff: Understanding (In-)Completeness of Proof Mechanisms for Separation Logic with Inductive Definitions

NETA ELAD, Tel Aviv University, Israel
ADITHYA MURALI, University of Wisconsin-Madison, USA
SHARON SHOHAM, Tel Aviv University, Israel

For over two decades Separation Logic has enjoyed its unique position as arguably the most popular framework for reasoning about heap-manipulating programs, as well as reasoning about shared resources and permissions. Separation Logic is often extended to include inductively-defined predicates, interpreted as least fixpoints, to form what is known as Separation Logic with Inductive Definitions (SLID). These inductive predicates are used to describe unbounded data-structures in the heap and to verify key properties thereof. Many theoretical and practical advances have been made in developing automated proof mechanisms for SLID, but by their very nature these mechanisms are imperfect, and a deeper understanding of their failures is desired. As expressive as Separation Logic is, it is not surprising that it is incomplete: there is no procedure that will provide a proof for all valid entailments in Separation Logic. In fact, at its very core, Separation Logic contains several sources of incompleteness that defy automated reasoning.

In this paper we study these sources of incompleteness and how they relate to failures of proof mechanisms of SLID. We contextualize SLID within a larger, relaxed logic, that we call Weak Separation Logic (WSL). We prove that unlike SLID, WSL enjoys completeness for a non-trivial fragment of quantified entailments with background theories and inductive definitions, via a reduction to first-order logic (FOL). Moreover, we show that the ubiquitous fold/unfold proof mechanism, which is unsurprisingly incomplete for SLID, does constitute a sound and complete proof mechanism of WSL, for theory-free, quantifier-free entailments with inductive definitions. In some sense, this shows that WSL is the natural logic of such proof mechanisms. Through this contextualization of SLID within WSL, we understand proof failures as stemming from rogue, nonstandard models, that exist within the class of models considered by WSL, but do not adhere to the stricter requirements of SLID. These rogue models are typically infinite, and we use the recently proposed formalism of symbolic structures to represent and automatically find them.

We present a prototype tool that implements the encoding of WSL to FOL and test it on an existing benchmark, which contains over 700 quantified entailment problems with inductive definitions, a third of which also contain background theories. Our tool is able to find counter-models to many of the examples, and we provide a partial taxonomy of the rogue models, shedding some light on real-world proof failures.

CCS Concepts: • **Theory of computation → Separation logic**; **Automated reasoning**; *Logic and verification.*

Additional Key Words and Phrases: proof mechanisms, nonstandard models, rogue models, infinite models

ACM Reference Format:
Neta Elad, Adithya Murali, and Sharon Shoham. 2026. Separating the Wheat from the Chaff: Understanding (In-)Completeness of Proof Mechanisms for Separation Logic with Inductive Definitions. *Proc. ACM Program. Lang.* 10, POPL, Article 29 (January 2026), 32 pages. https://doi.org/10.1145/3776671

Authors' Contact Information: Neta Elad, Tel Aviv University, Tel Aviv, Israel, netaelad@mail.tau.ac.il; Adithya Murali, University of Wisconsin-Madison, Madison, WI, USA, adithyamurali@cs.wisc.edu; Sharon Shoham, Tel Aviv University, Tel Aviv, Israel, sharon.shoham@gmail.com.

ACM 2475-1421/2026/1-ART29
https://doi.org/10.1145/3776671

1 Introduction

Automated techniques for program verification have made incredible advances in the last decade owing to the development of effective algorithms for checking logical validity and entailment. The most notable examples of this are SMT solvers [2, 3, 20], which implement decision procedures for validity of various combinations of quantifier-free decidable theories [53]. The development of theory and algorithms for decidable combinations of theories has been a critical foundation for the many successes of automated verification.

However, from the perspective of practice there remain significant gaps in our foundations. Practical program verification frequently requires the use of expressive logics for which decision procedures cannot exist; in fact, most of them are incomplete! Plainly, there cannot exist sound proof systems or validity procedures that can prove every valid formula. Furthermore, there are no canonical approaches for automating validity for such logics, and the theoretical limitations of existing automation techniques are not well-understood.

This lack of theoretical grounding comes at a practical cost. Verification tools implement heuristic approaches for reasoning with incomplete logics that can fail to prove valid formulas without clear reason. This is a particularly frustrating form of failure, since in practice it can be hard to distinguish plainly invalid formulas from valid formulas on which particular heuristics meet their limitations. Additionally, even savvy users who are willing to work through the failures can find systematic approaches for doing so lacking. This absence of 'diagnosability' breeds further frustration.

Thus, a foundational understanding of the incompleteness of validity/entailment checking mechanisms for various expressive logics is sorely needed.

1.1 Separation Logic with Inductive Definitions and Background Theories

Separation Logic (SL) is arguably the most popular logic for specifying and reasoning about heap-manipulating programs [21, 56, 57, 69]. Formulas are interpreted over a store s and a finite heap h, and can express a combination of *pure* properties that only involve the store as well as *spatial* properties that describe heaplets (portions of a heap) and separations between them. In practice, separation logic is augmented with inductively defined predicates to describe unbounded structures on the heap and their properties, e.g., linked lists, trees, sorted-ness, length/height, set of keys stored in the structure, etc. These predicates are interpreted as the least fixpoint of their definitions. The logic is also usually presented with a combination of background theories for reasoning about data values. We refer to this logic as SLID (Separation Logic with Inductive Definitions).

Unsurprisingly, SLID is incomplete. Formally, checking the validity of an entailment between SLID formulas is not recursively enumerable. There are many sources of incompleteness in SLID: (a) the magic wand, (b) the consideration of validity only over finite heaplets, (c) least fixpoints in the inductive definitions, (d) combinations with intended models of background sorts used for data reasoning, and (e) implicit second-order quantification over heaplets in the separating conjunction.[1]

Among these sources, it is widely agreed that the magic wand is difficult to reason with (it is essentially a form of second-order quantification [12]), and in fact many automated tools do not support it [71]. We therefore consider the automation of separation logics without the magic wand, and focus on the impact of the four latter sources of incompleteness.

[1]All of these sources except for the last one are well known causes of incompleteness in logics. Finite model theory is known to be incomplete for First-Order Logic, least fixpoints can define arbitrary computations, and logics that combine, say integers and uninterpreted functions can already express multiplication which yields incompleteness. We discuss these arguments elaborately in the context of separation logic in the extended version of the paper [24].

The Research Questions. There are several approaches to dealing with incompleteness in the literature. Some works focus on identifying decidable fragments of SLID, while others design new heap logics that enjoy certain completeness properties [5, 6, 36, 50].

In this work we seek a different pragmatic end: *Why* does existing automation for separation logic fail to be predictable? In other words, which among the many sources of incompleteness cause proof mechanisms to fail? What are the formal connections between the sources of incompleteness and the failure of proof for specific entailments, and more importantly, what does such a theory say about how to overcome proof failures?

We focus our investigation on one of the most popular techniques for automating validity checking of entailments in SLID: fold/unfold-based reasoning. Variants of this technique are implemented in several automated verifiers and entailment checkers for SLID [16, 49, 50, 68, 76].

Fold/Unfold Proof Mechanisms. The core idea in fold/unfold-based reasoning is the following: given an inductively defined predicate $P(\boldsymbol{x})$ with definition $\rho(\boldsymbol{x})$, these mechanisms *unfold* the body of the definition on predicate terms occurring in the entailment. More precisely, when a term $P(\boldsymbol{t})$ occurs in an entailment, they replace it with $\rho(\boldsymbol{t})$ and treat any further occurrences of P as an uninterpreted predicate. Similarly, these mechanisms may also *fold* a definition, replacing $\rho(\boldsymbol{t})$ with $P(\boldsymbol{t})$. They then try to prove the resulting entailment valid in SL (without inductive definitions) using a complete proof mechanism. This is clearly an overapproximation, in the sense that proving the unrolled entailment valid implies the validity of the original entailment. Of course, the converse may not hold in general.

Since SLID is incomplete, all sound proof mechanisms — including fold/unfold mechanisms — cannot prove every valid entailment. However, the limitations are not merely theoretical. These mechanisms do in fact fail to prove valid entailments that arise in practical program verification [23], and users can find themselves stuck not knowing what to do next. The lack of theoretical foundations to understand or deal with these failures is a key obstacle towards making practical and predictable automation for separation logic possible.

> *In this work we develop foundations for understanding the (in-)completeness of fold/unfold mechanisms for checking entailments in Separation Logic with Inductive Definitions.*

1.2 Logical Characterizations of (In-)Completeness and Rogue Models

We address the aforementioned research questions by developing *logical characterizations* of the (in-)completeness of fold/unfold proof mechanisms for separation logic. Our approach is inspired by recent works [46, 51] which pursue the following idea: Let $\mathcal{L}_1$ be a logic whose formulas are interpreted over a class of models $\mathcal{M}_1$, and let $\mathcal{S}$ be a proof mechanism which is incomplete for $\mathcal{L}_1$, i.e., it cannot prove every entailment that is valid in $\mathcal{L}_1$. The (in-)completeness of $\mathcal{S}$ is characterized by defining a logic $\mathcal{L}_2$ whose syntax (and satisfaction relation) is the same as that of $\mathcal{L}_1$, but where valid formulas (or entailments) must hold on a class of models $\mathcal{M}_2$ which is larger than $\mathcal{M}_1$. The technique then involves showing that $\mathcal{S}$ is a **sound proof mechanism** for entailments in $\mathcal{L}_2$. This means that the proving power of $\mathcal{S}$ is bounded by a *weaker* logic. Note that if an entailment is valid in $\mathcal{L}_2$, then it is certainly valid in $\mathcal{L}_1$, since $\mathcal{L}_2$ is interpreted over a larger class of models. However, the converse need not hold. Importantly, this means that entailments that hold in $\mathcal{L}_1$ but not in $\mathcal{L}_2$ *cannot* be proven using $\mathcal{S}$. This helps explain why $\mathcal{S}$ can fail to prove some entailments that are valid in $\mathcal{L}_1$: it is because these entailments may actually be invalid in $\mathcal{L}_2$, which is in some sense the "true logic" over which $\mathcal{S}$ operates.

The development of a logical characterization also yields an interesting way to witness the limitations of $\mathcal{S}$ in proving specific entailments. Suppose that an entailment $\alpha \models \beta$ is valid in $\mathcal{L}_1$ but invalid in $\mathcal{L}_2$. This means that there is a model M belonging to the class $\mathcal{M}_2$ such that M

satisfies α but not β. Note that M cannot belong to $\mathcal{M}_1$, since the entailment is valid in $\mathcal{L}_1$. Such a model M is referred to as a **rogue model**. The term rogue model refers to the fact that a user using $\mathcal{S}$ to prove entailments in $\mathcal{L}_1$ thinks about the validity of formulas over the intended or *standard* class of models $\mathcal{M}_1$, but their attempt is foiled by the *rogue* model M that lies outside this class. Additionally, if $\mathcal{S}$ is **complete** for $\mathcal{L}_2$, then it follows that *every* failure of $\mathcal{S}$ to prove an entailment valid in $\mathcal{L}_1$ can be traced to the existence of a rogue model. A logical characterization of the completeness thus helps not only understand and compare the relative completeness or incompleteness of proof mechanisms, but also yields a way to connect the incompleteness of $\mathcal{S}$ to its failure to prove specific entailments via rogue models.

We note here that the works we draw inspiration from do not explicitly develop the idea of a logical characterization of completeness. We distill the approach employed in these works into the above insight and apply the framework to separation logic. This is nontrivial, and is also the first such investigation of separation logic as prior works only studied first-order logics.

1.3 Main Contributions

The central insight of this work is that the (in-)completeness of fold/unfold mechanisms can be understood through the lens of a new separation logic that is constructed from standard SLID by ablating the various sources of incompleteness. By defining this weaker separation logic and studying its models, we are able to predict the failures of fold/unfold mechanisms on SLID entailments, connect the failures to different sources of incompleteness, and formulate systematic recommendations for overcoming them. These insights are the result of the following contributions:

1. A Weak Separation Logic (Section 4). Our first contribution is the development of a new separation logic to explore the influence of the various sources of incompleteness in standard SLID. We do this by systematically relaxing the various design decisions that lead to incompleteness in SLID. We remove the influence of least fixpoints by allowing interpretations for inductively defined predicate symbols to conform to any fixpoint of the definition. We remove the incompleteness arising from considering only finite heaplets by defining satisfaction of formulas over both finite and infinite heaplets and defining validity to mean satisfaction over both finite and infinite heaplets. We also ablate the incompleteness arising from requiring standard models of background theories by considering instead the class of all models of the theories (e.g., allowing nonstandard models of Presburger Arithmetic). We call the resulting logic **Weak Separation Logic** (WSL).

While it may be common to tacitly consider such relaxations when developing proof mechanisms for logics, our key insight is that we can perform the process in reverse: namely that we can characterize fold/unfold mechanisms by pursuing certain relaxations. This is rather subtle. In fact, while one may expect the above (perhaps usual) relaxations to be sufficient to eliminate all incompleteness, we show that performing any combination of the above relaxations — or indeed all of them — is not sufficient (see details in the extended version of the paper [24]).

The remaining source of incompleteness of WSL is the second-order quantification introduced by the separating conjunction: an SL formula $\alpha * \beta$ holds on a heaplet h if there exists a partitioning of h such that one partition satisfies α and the other satisfies β. Similarly, showing that the formula does not hold requires an argument for all possible partitionings. This weak form of second-order quantification turns out to be sufficient to yield incompleteness (see details in the extended version of the paper [24]). Further, it poses challenges for the relaxation of finite heaplets to finite and infinite heaplets, as infinite heaplets may have infinitely many partitionings.

We tackle this by first observing that many inductive definitions that arise in practice are well-behaved in a certain sense: in any particular global heap, there can be at most one heaplet that satisfies an inductive predicate. This is natural; after all, formulas are meant to capture specific

portions of memory. We then define a fragment of WSL, called the Effectively Determined-Heap (EDH) fragment, where a weaker form of this property is lifted to all formulas, ensuring that only finitely many heaplets must be considered for each formula. The combination of the relaxed semantics of WSL and the EDH restrictions allow us to establish completeness through a validity-preserving translation into first-order logic, generalizing similar existing encodings (e.g., [10]).

The FO translation is a significant contribution because it allows us to transfer established theoretical techniques for analyzing first-order logic and apply them to separation logics. In fact, it immediately yields that the EDH fragment of WSL admits complete validity checking for entailments! This is nontrivial since the fragment is quite rich, and in particular allows formulas to include some quantification over heap locations. In fact, it is undecidable. The connection between WSL and FOL is used crucially to analyze the completeness of fold/unfold mechanisms.

2. Soundness and Completeness of Fold/Unfold Reasoning for WSL (Section 5). Our second contribution is an analysis of the proving power of fold/unfold mechanisms for WSL. We formally model a large class of fold/unfold mechanisms and show these mechanisms are sound for WSL. This is an important result, because it establishes an upper bound on the power of the proof mechanisms. Entailments that are invalid in WSL cannot be proven by fold/unfold mechanisms, and therefore fold/unfold mechanisms are bound to fail for such entailments even when they are valid in SLID.

Completeness of fold/unfold reasoning for WSL is much more subtle to pose as it depends on the specific unfolding strategies, and variants can differ in proving power. Following the intuitive connection between fold/unfold operations and quantifier instantiation (employed in, e.g., [7, 48, 60]), our broad insight for proving completeness is that we can formalize this connection and establish a natural correspondence between fold/unfold transformations in WSL and quantifier instantiations in FOL. We utilize this to show that validity of entailments in quantifier-free theory-free WSL can always be proved by finitely many fold/unfold steps, and, thus, mechanisms that fold and unfold definitions on all terms are complete for this fragment of WSL. This result is a culmination of several developments, including the fact that WSL entailments admit a validity-preserving translation to FOL, the correspondence between fold/unfold axioms and quantifier instantiations, and properties of first-order logic including Herbrand's theorem and compactness. This is another instance where our contributions offer a way to use mathematical tools from the FOL literature in the context of separation logics.

Importantly, our soundness and completeness results show that WSL serves as a logical characterization of the completeness of fold/unfold mechanisms. Our results also imply that the failure of fold/unfold mechanisms to prove entailments in SLID without theories and quantifiers can be *entirely* explained by WSL. We believe that the evidence strongly points to a similar result for SLID with theories and (restricted) quantification as well, where fold/unfold applications are combined with other quantifier instantiations to handle first-order quantifiers and theories.

3. Symbolic Representation and Synthesis of Rogue Models (Section 6). We then turn to the connection between WSL and the failure of fold/unfold mechanisms on specific entailments. Our completeness results show that when fold/unfold mechanisms fail to prove an entailment in SLID, we can look for a rogue model on which the entailment, interpreted in WSL, does not hold. To find such rogue models we adopt the mathematical framework of symbolic structures developed in recent work [27], which defines a class of infinite models for first-order logic that admit finitary symbolic representations and efficient synthesis for refutation. Once again, we utilize the validity-preserving (and model-preserving) translation of WSL entailments to FOL.

We extend the theoretical developments in the prior work to utilize the algorithmic techniques in the context of separation logic. In particular, we extend symbolic structures with the background theory of the integers, synthesizing rogue models that are extensions of the standard model of the

integers for practical SLID entailments. We also identify a large class of inductive definitions called *heap-reducing* definitions for which rogue models, if they exist, must be infinite. To our knowledge, almost every inductive definition encountered in practice falls into this class. This substantiates our approach of representing rogue models using a framework for expressing infinite structures.

4. Empirical Evaluation of Rogue Models for SLID Entailments (Section 7). We implement a tool for checking validity of SLID entailments under WSL semantics via our translation to FOL. The tool dually attempts to (a) prove validity of entailments under WSL semantics (which would of course show validity in SLID), and (b) synthesize rogue models which would show that it is not valid in WSL semantics (and could be nontrivial to prove in SLID). Our synthesis algorithm builds on the model-finding procedure developed in [27], fine-tuning an abstract template to represent rogue models of practical SLID entailments (extended to our setting with the theory of the integers as described above). We evaluate our tool on a suite of over 700 entailments established in prior work [78] and show that it is effective at proving entailments as well as finding rogue models. We also analyze the rogue models qualitatively and provide a partial taxonomy. To the best of our knowledge, this is one of only two systematic studies of rogue models performed at scale [9], and the only one on rogue models for heap logics.

Proofs for the various claims and theorems throughout the paper are available in the extended version of the paper [24].

2 Overview

In this section we illustrate fold/unfold reasoning for Separation Logic with Inductive Definitions (SLID) and study its failures. We relate proof failures to a semantic gap between the intended semantics and the actual, relaxed semantics that fold/unfold captures. We demonstrate how rogue models, which arise from the semantic gap, can help understand the root cause of proof failure, and guide the user towards the missing parts of the proof.

2.1 Fold/Unfold Proof Mechanisms

Consider the following inductive definition of an acyclic list segment between two locations x and y, which starts from the base case where $x = y$ and extends the segment from the left, building on a segment between a location u that x points to and y. In the definition, the separating conjunction $*$ is used to require that x is not in the set of locations in the list segment between u and y.

$$\texttt{lseg}(x, y) \coloneqq x = y \vee (x \neq y * \exists u. x \mapsto u * \texttt{lseg}(u, y)) \tag{1}$$

For locations x and y in the heap, $\texttt{lseg}(x, y)$ characterizes the set of locations in the list segment between x and y, or, in other words, the unique heaplet of $\texttt{lseg}(x, y)$. The definition in Eq. (1) demonstrates a typical property of inductive definitions, where each tuple of locations that satisfy an inductive predicate has a unique, determined heaplet. In fact, this matches a core design goal of separation logic, where formulas are used to characterize heaplets. Accordingly, we only consider inductive definitions that induce determined heaplets. (As explained in Section 1, this restriction also turns out to be paramount for avoiding the incompleteness of SLID.)

One simple property of a list segment is that for any two distinct locations a, b that form a list segment, there exists a location v that a points to, such that v and b form a list segment. This is written in separation logic as the following entailment.

$$a \neq b * \texttt{lseg}(a, b) \vdash \exists v. a \mapsto v * \texttt{lseg}(v, b) \tag{2}$$

This entailment can be proved by unfolding $\texttt{lseg}(a, b)$ into its definition, resulting in

$$a \neq b * (a = b \vee (a \neq b * \exists u. a \mapsto u * \texttt{lseg}(u, b))) \vdash \exists v. a \mapsto v * \texttt{lseg}(v, b),$$

which is clearly valid, regardless of the interpretation of `lseg`. This example shows a successful application of a fold/unfold proof, which more generally may unfold any instance of an inductive predicate into its definition, or, conversely, fold its definition into an application of the inductive predicate. For example, the entailment

$$a \neq b * a \mapsto c * \texttt{lseg}(c, b) \vdash \texttt{lseg}(a, b) \tag{3}$$

can be proved by folding the antecedent into $\texttt{lseg}(a, b)$.

Unfortunately, fold/unfold reasoning may also fail to prove a valid entailment. For example, consider the following entailment, which states that a list segment can be extended from the right, i.e., obtaining a list segment between locations a and b from a list segment between a and some location c that points to b which is not yet in the list (ensured by a separating conjunction applied to $b \mapsto nil$)

$$\texttt{lseg}(a, c) * c \mapsto b * b \mapsto nil \vdash \texttt{lseg}(a, b) * b \mapsto nil \tag{4}$$

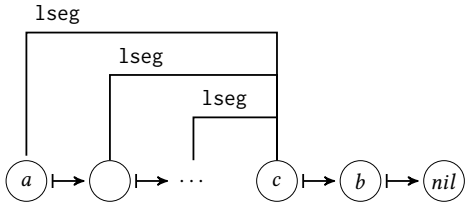
This entailment is also valid, but cannot be proved with any number of fold/unfold steps. Intuitively, the syntactic shape of the definition of `lseg` restricts fold/unfold reasoning to the left side of a segment, but here validity hinges on the points-to relation $c \mapsto b$ between locations on the right side of segments. Therefore, no matter how many fold/unfold steps are performed, we will never be able to derive $\texttt{lseg}(x, b)$ in conjunction with $a \mapsto^* x$, which is required for ultimately deriving $\texttt{lseg}(a, b)$. Beyond this technical explanation, the failure to prove Eq. (4) can be understood on a deeper level as a mismatch between the intended semantics and the semantics captured by fold/unfold. In the intended semantics, interpretations of `lseg` are least fixpoints, hence all list segments are finite, but fold/unfold proofs cannot distinguish least fixpoints from other fixpoints, which also include infinite list segments. Unfortunately, when potentially infinite fixpoint interpretations of list segments are considered, the symmetry between the left and right sides of a segment, which exists in finite segments, is broken. As a result, list segments constructed by adding locations on the left (as defined by `lseg`) cannot necessarily be constructed by adding locations on the right. Thus, the entailment in Eq. (4) is not valid over such interpretations. As we will see in the sequel, this abstract train of thought can be reified.

2.2 Weak Separation Logic

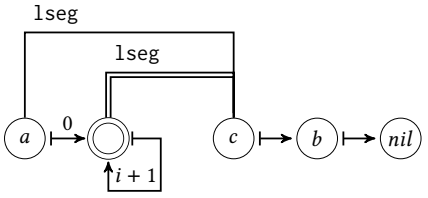
As illustrated by the failed attempt to prove the entailment in Eq. (4), fold/unfold proofs only capture the fixpoint nature of interpretations, and consider both finite and infinite heaps. We refer to SLID with this relaxed semantics as *Weak Separation Logic* (WSL), formally defined in Section 4. When theories are involved, WSL also relaxes the standard interpretation of the theory to any interpretation satisfying its first-order axiomatization. As a result, the class of models defining the semantics of WSL is a strict super-set of the class defining SLID. As we show in Section 5, WSL is in fact the logic captured by fold/unfold. Namely, the fold/unfold proof mechanism is both sound and complete for the theory-free, quantifier-free fragment of WSL.

The choice of the weak semantics of WSL is not arbitrary: as discussed in Section 1, the relaxations that culminate in WSL correspond exactly to the sources of incompleteness of SLID. Surprisingly, these very same relaxations lead to a precise logical characterization of fold/unfold proof mechanisms.

The theory-free, quantifier-free fragment of WSL is complete despite the remaining implicit second-order quantification, which is generally a source of incompleteness. The panacea lies in the restrictions of the fragment, which in conjunction with the uniqueness of heaplets of inductive definitions, guarantee that only finitely many partitions of heaplets must be considered when checking entailments. We use the same idea to generalize the completeness result to the larger Effectively Determined-Heap (EDH) fragment of WSL, where this property enables a reduction to FOL and paves the way to completeness. As we will see in the sequel, the reduction to FOL is not



(a) The location a forms a list segment with c, justified by the infinitely many list segments that end in c. But a does not form a list segment with b despite the fact that c points to b, since none of the locations reachable from a (through the points-to relation) forms a list segment with b. One can easily verify that this is a fixpoint interpretation of Eq. (1).



(b) A symbolic structure representing the same rogue model in Fig. 1a. The infinitely many heap locations between a and c are represented succinctly by the double-circled node in the middle, where each location corresponds to an index $i \geq 0$. The points-to relation connects i to $i+1$, and connects a to index 0.

Fig. 1. A rogue counter-model to the entailment of Eq. (4), depicted both explicitly (on the left) and as a symbolic structure (on the right). Locations in the heap are shown as circles, $\mapsto$ arrows represent the points-to relation. The `lseg` lines show the relevant list segments ending with c, others omitted to avoid clutter. Formally, the points-to relation is defined as $\{a \mapsto 0, c \mapsto b, b \mapsto nil\} \cup \{\, i \mapsto i+1 \mid i \geq 0 \,\}$, and the list segment relation is defined as $\{\langle a,a\rangle, \langle a,c\rangle, \langle c,c\rangle, \langle c,b\rangle, \langle c,nil\rangle\langle b,b\rangle, \langle b,nil\rangle\} \cup \{\, \langle a,i\rangle \mid i \geq 0 \,\} \cup \{\, \langle i,c\rangle \mid i \geq 0 \,\} \cup \{\, \langle i,j\rangle \mid 0 \leq i \leq j \,\}$.

just a means for proving completeness of the EDH fragment of WSL, but also a cornerstone in our approach for investigating proof failures of fold/unfold mechanisms.

2.3 Rogue Models

Continuing with our study of fold/unfold proof mechanisms, we realize that since WSL provides a logical characterization of fold/unfold reasoning, fold/unfold proofs are made against the models of WSL. Circling back to the entailment of Eq. (4), we understand that the failure of fold/unfold proofs is due to the gap between the class of models considered by WSL and the intended class of models of SLID. This semantic gap gives rise to entailment-refuting *rogue counter-models*, that demonstrate the root cause of failure.

A depiction of one such counter-model is given in Fig. 1a. The heap of this rogue model consists of infinitely many locations, where a is the head of an infinite sequence of locations, each connected to the next by the points-to relation. The sequence never reaches c, but nonetheless all locations in it form a list segment with c, in particular $\texttt{lseg}(a, c)$ is true. To simplify the diagram, these are the only `lseg` lines depicted, but $\texttt{lseg}(x, y)$ is also true for every pair of locations in the segment between a and c, where x precedes y in the sequence. Other than that, the only non-reflexive segment is between c and b. We can see that this is indeed a valid fixpoint interpretation of `lseg`, although not a least fixpoint (nor a greatest fixpoint). For example, $\texttt{lseg}(a, c)$ is justified by the list segment between the successor of a and c, which in turn is justified by the list segment between the successor of the successor of a and c, and so forth. Since in addition c points to b, but $\texttt{lseg}(a, b)$ is false, it follows that the antecedent of entailment in Eq. (4) holds, but the consequent does not. This a rogue counter-model, since it is neither finite nor interprets `lseg` as a least fixpoint.

Rogue counter-models serve as a proof of unprovability of entailments using fold/unfold, and hint that reasoning beyond fold/unfold is necessary. Here, the existence of an infinite list segment between a and c hints at a need for inductive reasoning over the definition of `lseg`. Indeed, the entailment of Eq. (4) is inductively provable, which means that it is provable in the base case where

$a = c$, and assuming the entailment for some list segment $\texttt{lseg}(u, c)$ such that $a \mapsto u$, it is also provable for the list segment $\texttt{lseg}(a, c)$, constructed according to the inductive case of the definition. Both the base case and the inductive step can be proved by fold/unfold proof mechanisms, meaning that the only missing piece to the proof was a single application of the induction principle, as indicated by the rogue model.

In order for us to leverage rogue models for understanding and fixing proof failures, we need a way to represent and find them. This task is challenging, since, as seen in Fig. 1a, they may be infinite. In fact, for a large class of so-called "heap-reducing" inductive definitions (which includes `lseg`), rogue models must be infinite, as we prove in Section 6. Fortunately, we have two helpful tools at our disposal. The first is the FOL encoding of WSL, which allows us to reduce the problem of finding rogue counter-models of SLID entailments to finding satisfying first-order structures for the FO-encoded entailment. The second is the recently introduced formalism of symbolic structures [27], which are finite representations of possibly infinite first-order structures. Symbolic structures enjoy decidable model-checking and are amenable to automatic (though unsurprisingly incomplete) model-finding, which, through the FO encoding, we utilize to find rogue models.

The rogue model of Fig. 1a can be represented by the symbolic structure shown in Fig. 1b (making the same omissions of `lseg` lines). The symbolic structure compacts the infinitely many heap locations in the segment between a and c into a single symbolic *summary node*, which represents each heap location with an integer $i \geq 0$, called an index. The points-to relation of each location i represented by the summary node is given by $i + 1$, meaning it points to another location of that same summary node that has the index $i + 1$. The points-to relation relates a to index 0 within the summary node. The `lseg` predicate treats the summary node as a whole, interpreting all locations within it as forming a list segment with c.

Summary nodes give us the ability to pinpoint the core "rogueness" of the model, and hint more directly at how induction should be used. Though in the case of Eq. (4) the entailment itself was inductive, in other cases, induction is an intermediate step, when an inductive lemma is used as part of a larger proof. For example, consider the following slight variation of Eq. (4), where we have the added assumption that $a \neq b$:

$$a \neq b * \texttt{lseg}(a, c) * c \mapsto b * b \mapsto nil \vdash \texttt{lseg}(a, b) * b \mapsto nil \tag{5}$$

The entailment of Eq. (5) is not inductive by itself, but follows (by fold/unfold reasoning) from the inductively-provable entailment of Eq. (4). The rogue model of Fig. 1 is also a counter-model for Eq. (5), and helps identify Eq. (4) as a key lemma for completing the proof. Firstly, it guides the user away from unhelpful lemmas and towards useful ones: any entailment that already holds in the counter-model would not help us eliminate it, and thus we can focus on entailments not satisfied by the rogue model. I.e., entailments where the antecedent holds in the model but the consequent does not. In the case of Fig. 1, one can see that the entailment $\texttt{lseg}(a, c) * c \mapsto b * b \mapsto nil \vdash \texttt{lseg}(a, b) * b \mapsto nil$ of Eq. (4) is one such possibility, since all the atoms in the antecedent are satisfied but at least one of the atoms in the consequent is not. Secondly, the rogue model also hints where induction should be used. We can see that though $\texttt{lseg}(a, c)$ is true in the rogue model, the nodes of a and c are not connected in the symbolic structure according to the points-to relation. This means that $\texttt{lseg}(a, c)$ cannot be justified in a least-fixpoint interpretation, and thus induction on this term could prove useful. These two sources of guidance can be used somewhat systematically: mapping out formulas that are satisfied and not satisfied by the model to construct lemmas, and looking for inductive-predicate atoms that "close over" infinite parts of the model (that manifest as disconnected in the symbolic representation) to inform induction.

For a more elaborate example consider a system of inductive definitions that uses the background theories of integers and sets to model (strictly) sorted lists and their keys, where we have two inductive predicates $\texttt{slist}(x)$, for sorted list, and $\texttt{keys}(x, Y)$, for the set of keys in elements

reachable from x:

$$\begin{aligned}\texttt{slist}(x) &\coloneqq x = nil \vee (\exists u. x \mapsto u * \texttt{slist}(u) * \texttt{key}(x) < \texttt{key}(u))\\ \texttt{keys}(x, Y) &\coloneqq (x = nil \wedge Y = \emptyset) \vee (\exists u, Z. x \mapsto u * \texttt{keys}(u, Z) * Y = Z \cup \{\texttt{key}(x)\}),\end{aligned}$$

and the following entailment:

$$a \mapsto b * \texttt{key}(a) < \texttt{key}(b) * (\texttt{slist}(b) \wedge \texttt{keys}(b, K)) \vdash \texttt{slist}(a) * \texttt{key}(a) \notin K.$$

The entailment is valid in SLID but not in WSL since WSL allows (rogue) models where b is the head of an infinite sorted list and $\texttt{key}(a)$ is in the set of keys reachable from it even though $\texttt{key}(a) < \texttt{key}(b)$. For example, if we identify each element in the heap with its key, one such rogue model can be described with the following points-to relation: $a = 0 \mapsto b = 1 \mapsto 2 \mapsto \ldots$; where $\texttt{slist}(i)$ is true for all i; and $\texttt{keys}(i, K)$ is true iff $K = \{0, i, i+1, i+2, \ldots\}$. Looking at this model, it is clear that for any $\texttt{keys}(i, K)$ where $i > 0$, K spuriously contains the element 0. This interpretation of `slist` and `keys` is a valid fixpoint but not a least fixpoint, where a key is only included in the set of reachable keys if a corresponding element is in the list. Since the keys keep increasing in a sorted list, this ensures that in a least-fixpoint interpretation we will never encounter a key smaller than the key of the head of the list. In particular, in any least-fixpoint interpretation where $\texttt{key}(a) < \texttt{key}(b)$, $\texttt{slist}(b)$ and $\texttt{keys}(b, K)$ hold, we expect $\texttt{key}(a) < k$ for any $k \in K$. This property, which is violated in the rogue model, can be written as an entailment that is provable by induction on $\texttt{slist}(b)$ and implies the desired property. The rogue model not only helps identify this entailment as an auxiliary lemma, but also hints at induction on $\texttt{slist}(b)$ for proving it since b is disconnected from *nil* (the base case of `slist`). Representing this kind of rogue model, where multiple background theories are involved, requires careful modeling, but is possible in theory.

The use of symbolic structures to represent rogue models is not merely a theoretical curiosity. As we discuss in Section 7, we developed a prototype tool that implements the FO encoding of WSL with the background theory of linear integer arithmetic, and uses FEST [27] to find symbolic rogue counter-models. In parallel, the tool uses Z3 [20] to check the validity of the FOL formulas resulting from the encoding, which implies validity in WSL and therefore in SLID. We evaluated the effectiveness of this approach over a benchmark suite consisting of over 700 examples of entailments with inductive definitions, taken from the SLS tool for inductive lemma synthesis in SLID [78]. Our tool is able to find infinite rogue models for up to 92% of the examples that require reasoning beyond fold/unfold, and provide helpful feedback for understanding proof failures.

3 Preliminaries

First-Order Logic. We define many-sorted first-order logic (FOL) with equality in the usual way. A first-order vocabulary consists of sorted constant, function and relation symbols. We denote the set of logical variables $x, y, x', x_1, \ldots$ by $\mathcal{X}$, and assume each variable has an associated sort, omitted when clear from context. Terms are constructed from constants, variables and well-sorted function applications. Formulas are built up from equalities and relation applications using logical connectives $\neg, \vee, \wedge, \rightarrow$ and quantifiers $\forall, \exists$.

Background theories and LIA. In literature, background theories are usually defined either as theories of a standard model or by a set of axioms. We consider these two faces together, and accordingly define a theory as a triple $\mathcal{T} = (\Sigma^{\mathcal{T}}, M^{\mathcal{T}}, \Gamma^{\mathcal{T}})$ where $\Sigma^{\mathcal{T}}$ is a first-order vocabulary, $M^{\mathcal{T}}$ is a first-order structure for $\Sigma^{\mathcal{T}}$, and $\Gamma^{\mathcal{T}}$ is a recursive set of first-order sentences over $\Sigma^{\mathcal{T}}$ that is sound for $M^{\mathcal{T}}$ (i.e., $M^{\mathcal{T}} \models \Gamma^{\mathcal{T}}$). We call $\Sigma^{\mathcal{T}}$ the theory vocabulary, $M^{\mathcal{T}}$ the *standard model* of $\mathcal{T}$, and $\Gamma^{\mathcal{T}}$ the first-order axiomatization of $\mathcal{T}$. In particular we consider the theory of Linear Integer Arithmetic $\text{LIA} = (\Sigma^{\text{LIA}}, M^{\text{LIA}}, \Gamma^{\text{LIA}})$, where $\Sigma^{\text{LIA}} = \{0, 1, +, <\}$ over sorts $\{\texttt{int}\}$, M^{LIA} is the standard

model of the integers, and Γ^{LIA} is the complete first-order axiomatization of LIA. We denote the set of quantifier-free LIA formulas by QF_{LIA}.

4 A Weak Separation Logic

In this section we formally define Weak Separation Logic (WSL), a relaxed version of separation logic with inductive definitions and theories, that tackles its sources of incompleteness. For brevity, from now on we use SL also when referring to SLID, and let context clarify whether inductive definitions are involved. We establish completeness for a fragment of WSL by reduction to first-order logic. In Section 5 we further show that WSL captures the actual semantics that fold/unfold proof mechanisms for SL are sound and complete for. The class of satisfying WSL models is a strict superset of the class of satisfying SL models, giving rise to *rogue, nonstandard models*, which we discuss in Section 6.

4.1 Separation Logic with Inductive Definitions and Theories

We start by giving a model-oriented definition of SL. This allows us to analyze both SL and WSL through their classes of models, thereby getting better understanding of the semantic gap between them. In literature, the semantics of SL is usually given in relation to a store s and a heap h that together describe a program state. Our model-oriented description uses different objects, namely heap structures M, assignments v and heaplets η, to describe a state, but the two representations and corresponding semantics are equivalent.

From this point on, and throughout the paper, we fix a background theory $\mathcal{T} = \left(\Sigma^{\mathcal{T}}, M^{\mathcal{T}}, \Gamma^{\mathcal{T}}\right)$ over a set of sorts S. We start with the basic vocabulary and syntax of SL formulas.

Definition 4.1 (Vocabulary of Separation Logic). A vocabulary of SL over background theory $\mathcal{T}$ is a set $\Sigma = \Sigma^{\text{SL}} \uplus \Sigma^{\mathcal{T}}$, where $\Sigma^{\mathcal{T}}$ is the background vocabulary over sorts S, and Σ^{SL} is a *relational* first-order vocabulary over $S \uplus \{\boldsymbol{s}^*\}$ (i.e., containing only constant and relation symbols), required to contain a constant symbol *nil* of sort $\boldsymbol{s}^*$. We call $\boldsymbol{s}^*$ the *location sort*, and the predicates in Σ^{SL} the *inductive predicates*. By convention, we use $P, P', P_1, \ldots$ for inductive predicates and $Q, Q', Q_1, \ldots$ for theory predicates.

The syntax of SL formulas contains an atomic "points-to" relation, which relates locations to their contents. In order to simplify the presentation, and in particular remove uninteresting details from the FOL reduction, we assume a uniform, global, *record shape*, a tuple of sorts $\langle \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n \rangle$ that every location adheres to. This assumption is purely for presentation reasons, and in fact all results hold just the same for a set of record shapes.

Definition 4.2 (Syntax of SL formulas). Given a vocabulary $\Sigma = \Sigma^{\text{SL}} \uplus \Sigma^{\mathcal{T}}$ and a record shape $\langle \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n \rangle$ over $S \uplus \{\boldsymbol{s}^*\}$, terms and formulas are defined recursively, by the following grammar:

$$
\begin{aligned}
t &::= c \in \Sigma \mid x \in \mathcal{X} \mid f(t_1, \ldots, t_k) \\
\varphi &::= t_1 = t_2 \mid t_1 \neq t_2 \mid Q(t_1, \ldots, t_k) \mid \neg Q(t_1, \ldots, t_k) \mid \mathit{emp} \mid t \mapsto \langle t_1, \ldots, t_n \rangle \mid P(t_1, \ldots, t_k) \\
&\quad \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 * \varphi_2 \mid \exists x.\varphi \mid \forall x.\varphi
\end{aligned}
$$

We use the names *theory-free* for formulas where no theory terms appear, *quantifier-free* for formulas with no quantifiers, and *conjunctive* for formulas without disjunctions (but with both regular and separating conjunctions).

Though under standard SL semantics we are interested only in structures that interpret the inductive predicates as a least-fixed-point solution of a system of inductive definitions, in order to define this formally, we first define a satisfaction relation for triples of structure M, assignment v and heaplet η, then restrict our attention to valid LFP structures. Such a triple M, v, η is simply a

$$
\begin{array}{llll}
M,v,\eta & \models & t_1 \bowtie t_2 & \text{if } \bar{v}(t_1) \bowtie \bar{v}(t_2) \text{ and } \eta = \emptyset \\
M,v,\eta & \models & Q(t_1,\ldots,t_k) & \text{if } \langle \bar{v}(t_1),\ldots,\bar{v}(t_k)\rangle \in Q^M \text{ and } \eta = \emptyset \\
M,v,\eta & \models & \neg Q(t_1,\ldots,t_k) & \text{if } \langle \bar{v}(t_1),\ldots,\bar{v}(t_k)\rangle \notin Q^M \text{ and } \eta = \emptyset \\
M,v,\eta & \models & emp & \text{if } \eta = \emptyset \\
M,v,\eta & \models & P(t_1,\ldots,t_k) & \text{if } \langle \bar{v}(t_1),\ldots,\bar{v}(t_k),\eta\rangle \in P^M \\
M,v,\eta & \models & t \mapsto \langle t_1,\ldots,t_n\rangle & \text{if } \bar{v}(t) \neq \texttt{null}, h^M(\bar{v}(t)) = \langle \bar{v}(t_1),\ldots,\bar{v}(t_n)\rangle \text{ and } \eta = \{\bar{v}(t)\} \\
M,v,\eta & \models & \varphi_1 \vee \varphi_2 & \text{if } M,v,\eta \models \varphi_1 \text{ or } M,v,\eta \models \varphi_2 \\
M,v,\eta & \models & \varphi_1 \wedge \varphi_2 & \text{if } M,v,\eta \models \varphi_1 \text{ and } M,v,\eta \models \varphi_2 \\
M,v,\eta & \models & \varphi_1 * \varphi_2 & \text{if } \exists \eta_1,\eta_2.\ M,v,\eta_1 \models \varphi_1, M,v,\eta_2 \models \varphi_2 \text{ and } \eta = \eta_1 \uplus \eta_2 \\
M,v,\eta & \models & Qx\colon \boldsymbol{s}.\varphi & \text{if } Qd \in \mathcal{D}^M(\boldsymbol{s}).M,v[d/x],\eta \models \varphi
\end{array}
$$

Fig. 2. Recursive definition of satisfaction in Separation Logic. The $\bowtie$ symbol stands for either $=$ or $\neq$, the Q symbol stands for either $\forall$ or $\exists$, and $\eta = \eta_1 \uplus \eta_2$ is shorthand for $\eta = \eta_1 \cup \eta_2 \wedge \eta_1 \cap \eta_2 = \emptyset$.

different way of representing the typical store and heap of SL semantics. The store is split between interpretations of constants c^M and assignment to variables $v(x)$, the heap is split between a global heap mapping h^M that represents the heap's points-to relation and the heaplet η that describes the heap's domain, i.e., the set of allocated locations. The heap can therefore be reconstructed from h^M projected onto η. The satisfaction relation defined in Def. 4.4 then coincides with satisfaction by a store-heap pair.

Definition 4.3 (Heap structures). Given a vocabulary $\Sigma = \Sigma^{\text{SL}} \uplus \Sigma^{\mathcal{T}}$ and a record shape $\langle \boldsymbol{s}_1,\ldots,\boldsymbol{s}_n\rangle$ over $S \uplus \{\boldsymbol{s}^*\}$, a *heap structure* for $\Sigma, \langle \boldsymbol{s}_1,\ldots,\boldsymbol{s}_n\rangle$ is a triple $M = (\mathcal{D}, h, \mathcal{I})$, where $\mathcal{D}$ is a mapping of sorts to disjoint domains, of which $\mathcal{D}(\boldsymbol{s}^*)$, called a *location domain*, contains a designated element $\texttt{null} \in \mathcal{D}(\boldsymbol{s}^*)$; h is a *heap*, a mapping of non-null locations to records of domain elements, $h\colon \mathcal{D}(\boldsymbol{s}^*)_+ \to \mathcal{D}(\boldsymbol{s}_1) \times \cdots \times \mathcal{D}(\boldsymbol{s}_n)$, where $\mathcal{D}(\boldsymbol{s}^*)_+ \triangleq \mathcal{D}(\boldsymbol{s}^*) \setminus \{\texttt{null}\}$; and $\mathcal{I}$ is an *interpretation* of vocabulary symbols, defined in the usual way, except that for an inductive predicate symbol $P\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$, the interpretation is defined as $\mathcal{I}(P) \subseteq \mathcal{D}_P \triangleq \mathcal{D}(\boldsymbol{s}_1) \times \cdots \times \mathcal{D}(\boldsymbol{s}_k) \times \mathcal{P}(\mathcal{D}(\boldsymbol{s}^*))$. Additionally, we require that $\mathcal{I}(nil) = \texttt{null}$. For any symbol s, we denote $s^M = \mathcal{I}(s)$.

A structure M is called *heap-finite* if its location domain $\mathcal{D}(\boldsymbol{s}^*)$ is finite. We define a heaplet η as a set of non-null locations, i.e., $\eta \subseteq \mathcal{D}(\boldsymbol{s}^*)_+$, and an assignment as a well-sorted mapping $v\colon \mathcal{X} \to \mathcal{D}$ and we denote by $\bar{v}$ the lifting of v to any term t in the usual way.

Definition 4.4 (Satisfaction of SL formulas). Given a heap structure M, an assignment v and a heaplet η, we recursively define satisfaction of a SL formula in Fig. 2.

Using the definitions of heap structures and formula satisfaction, we now move to formally define systems of inductive definitions and their semantics.

Definition 4.5 (System of inductive definitions). Given a vocabulary Σ, a *system of inductive definitions* (SID) Φ is a mapping of each inductive predicate $P\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$, to a formula $\Phi(P)$ with free variables $\boldsymbol{x} = x_1,\ldots,x_k$, of the form $\Phi(P)(\boldsymbol{x}) = \exists \boldsymbol{y}.\ \bigvee_j \rho_j^P(\boldsymbol{x},\boldsymbol{y})$, where ρ_j^P is a quantifier-free, conjunctive SL formula.

Note that in literature, the inductive cases ρ_j^P are commonly defined as containing the existential quantifier. This is equivalent (under variable renaming), but we prefer this form for the simplicity of presentation.

Notation. Given a structure M over a vocabulary with inductive predicates $\boldsymbol{P} = \langle P_1,\ldots,P_m\rangle$, and a tuple of predicate interpretations $\boldsymbol{I} = \langle I_1,\ldots,I_m\rangle$, where $I_i \subseteq \mathcal{D}_{P_i}$, we denote the reinterpretation model $M' = M[\boldsymbol{I}/\boldsymbol{P}]$ where $P_i^{M'} = I_i$.

Definition 4.6 (Transformer for system of inductive definitions). Given a structure M over a vocabulary with inductive predicates $P_1, \ldots, P_m$, and a system of inductive definitions Φ, for each relation $P\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$, we define an operator $\Phi_P^M\colon \mathcal{D}_{P_1} \times \cdots \times \mathcal{D}_{P_n} \to \mathcal{D}_P$:

$$\Phi_P^M(\boldsymbol{I}) \triangleq \left\{ \langle d_1, \ldots, d_k, \eta \rangle \;\middle|\; \begin{array}{c} d_i \in \mathcal{D}^M(\boldsymbol{s}_i), \\ \eta \subseteq \mathcal{D}^M(\boldsymbol{s}^*), \\ M[\boldsymbol{I}/\boldsymbol{P}], [\boldsymbol{d}/\boldsymbol{x}], \eta \models \Phi(R) \end{array} \right\};$$

and we define a transformer for the entire system: $\Phi^M\colon \mathcal{D}_{P_1} \times \cdots \times \mathcal{D}_{P_n} \to \mathcal{D}_{P_1} \times \cdots \times \mathcal{D}_{P_n}$:

$$\Phi^M(\boldsymbol{I}) \triangleq \left\langle \Phi_{P_1}^M(\boldsymbol{I}), \ldots, \Phi_{P_n}^M(\boldsymbol{I}) \right\rangle.$$

Note that $\Phi^M(\boldsymbol{I})$ is monotone with respect to set inclusion (since all inductive predicates appear positively), and therefore has a least fixpoint [79]. We say that M is a FP structure for Φ when $\boldsymbol{P}^M = \Phi^M(\boldsymbol{P}^M)$ and that M is a LFP structure for Φ when $\boldsymbol{P}^M$ is the least set such that $\boldsymbol{P}^M = \Phi^M(\boldsymbol{P}^M)$. We denote the classes of all FP, LFP structures for Φ as $\mathcal{M}_{\text{FP}}(\Phi), \mathcal{M}_{\text{LFP}}(\Phi)$ respectively.

Given a heap structure M and $\langle d_1, \ldots, d_k, \eta \rangle \in P^M$, we refer to η as a supporting heaplet of $\langle d_1, \ldots, d_k \rangle$. In general, a LFP or FP structure allows the same tuple of elements to have multiple supporting heaplets for the same inductive predicate. However, as explained in Section 1, this is both an obstacle for overcoming incompleteness resulting from second-order quantification over heaplets, and defies the design goal of separation logic, where formulas are meant to characterize heap locations. Indeed, typical SIDs ensure that in their LFP structures, every tuple of elements that satisfies an inductive predicate has a unique heaplet (often called footprint) that supports it. Next we formalize this property of structures, and restrict our attention to systems that guarantee it.

Definition 4.7 (Determined-heap structure). A structure M has a *determined heap* if for any inductive predicate $P\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$ and for any tuple of elements $d_1, \ldots, d_k$ there exists at most one heaplet $\eta \subseteq \mathcal{D}(\boldsymbol{s}^*)$ s.t. $\langle d_1, \ldots, d_k, \eta \rangle \in P^M$. For a determined-heap structure we may denote the interpretation of inductive predicate as a partial function $P^M\colon \mathcal{D}(\boldsymbol{s}_1) \times \cdots \times \mathcal{D}(\boldsymbol{s}_k) \rightharpoonup \mathcal{P}(\mathcal{D}(\boldsymbol{s}^*))$, mapping tuples of elements to their supporting heaplet.

We round this section off by summarizing the syntax and semantics of SL, and defining two kinds of entailment in SL: plain, and modulo a system of inductive definitions.

Definition 4.8 (Separation Logic). We define Separation Logic (SL) as a logic with the syntax of formulas defined in Def. 4.2, the syntax of inductive definitions defined in Def. 4.5, and semantics given as the satisfaction relation Def. 4.4, over the class $\mathcal{M}_{\text{SL}}$ of *heap-finite* determined-heap structures (Def. 4.7), that are extensions of the standard model of the background theory $M^{\mathcal{T}}$.

For a finite set of SL formulas Γ and a SL formula ψ, we say that the SL entailment problem for Γ and ψ, denoted $\Gamma \vdash_{\text{SL}} \psi$, is valid when for every model $M \in \mathcal{M}_{\text{SL}}$, assignment v and heaplet η, if $M, v, \eta \models \Gamma$ then $M, v, \eta \models \psi$. Further, we say that the SL entailment problem for Γ and ψ modulo SID Φ, denoted $\Gamma \vdash_{\text{SL}}^{\Phi} \psi$, is valid when for every model $M \in \mathcal{M}_{\text{SL}} \cap \mathcal{M}_{\text{LFP}}(\Phi)$, assignment v and heaplet η, if $M, v, \eta \models \Gamma$ then $M, v, \eta \models \psi$.

4.2 From Separation Logic to Weak Separation Logic

As discussed in Section 2, SL contains multiple sources of incompleteness, which makes it difficult to discern the root cause of a proof failure, be it a deficient proof or an unprovable entailment. In this section we define Weak Separation Logic (WSL), that relaxes the semantics of SL and allows us to understand proof failures of SL as stemming from the semantic gap between SL and WSL.

WSL tackles three of the four sources of incompleteness in SL with three relaxations that extend the class of structures considered: (i) structures are not necessarily extensions of the standard

model of the theory, but only required to satisfy its axiomatization; (ii) the location domain may be infinite (not just finite); and (iii) for entailments modulo systems of inductive definitions, we use FP structures (not just LFP structures). Formally:

Definition 4.9. We define Weak Separation Logic (WSL) with the same syntax of formulas and inductive definitions as in SL, and semantics given as the same satisfaction relation as in SL, but over the class $\mathcal{M}_{\text{WSL}}$ of *heap-finite or infinite* determined-heap structures that *satisfy* $\Gamma^{\mathcal{T}}$.

For a finite set of SL formulas Γ and SL formula ψ, we say that the WSL entailment problem for Γ and ψ, denoted $\Gamma \vdash_{\text{WSL}} \psi$, is valid when for every model $M \in \mathcal{M}_{\text{WSL}}$, assignment v and heaplet η, if $M, v, \eta \models \Gamma$ then $M, v, \eta \models \psi$. Further, we say that the WSL entailment problem for Γ and ψ modulo SID Φ, denoted $\Gamma \vdash^{\Phi}_{\text{WSL}} \psi$, is valid when for every model $M \in \mathcal{M}_{\text{WSL}} \cap \mathcal{M}_{\text{FP}}(\Phi)$, assignment v and heaplet η, if $M, v, \eta \models \Gamma$ then $M, v, \eta \models \psi$.

Recall that the remaining source of incompleteness in SL is the implicit second-order quantification over heaplets. WSL does not tackle this source directly, but we later consider a fragment of WSL, where restricted quantification together with determined-heap property of structures allow us to overcome this obstacle.

We now focus our discussion on the relationship between SL and WSL. WSL is clearly a relaxation of SL, in the sense entailments that are valid in WSL are also valid in SL, whether plain or modulo some SID, but the converse does not necessarily hold. However, next we show that when restricting ourselves to entailments where SL does not exhibit the sources of incompleteness, SL and WSL coincide, which means that in these cases the relaxations are inconsequential.

Intuitively, to eliminate incompleteness stemming from the theory, we restrict ourselves to the sub-logics of SL and WSL without theories. To remove the distinction between finite and infinite models and handle second-order quantification over heaplets, we restrict ourselves to QF formulas, which enjoy a finite-model property. Together these restrictions ensure that for every QF formula φ, there exist $M \in \mathcal{M}_{\text{SL}}$, assignment v and heaplet η s.t. $M, v, \eta \models \varphi$ iff there exist $M' \in \mathcal{M}_{\text{WSL}}$, v' and η' s.t. $M', v', \eta' \models \varphi$. Finally, to eliminate incompleteness stemming from the gap between LFP and FP semantics, we consider plain entailments, i.e., without systems of inductive definitions. Indeed we show that under these restrictions the semantic gap between SL and WSL is eliminated.

Claim 4.10. *Given a finite set of theory-free, quantifier-free formulas* Γ *and a theory-free, quantifier-free formula* ψ*, the entailment* $\Gamma \vdash_{\text{SL}} \psi$ *is valid iff* $\Gamma \vdash_{\text{WSL}} \psi$ *is valid.*

Claim 4.10 implies that proof systems for SL and WSL coincide under these restrictions, and therefore the logics are "as complete" in this setting. In fact, it is not difficult to see that both logics are complete under these restrictions since they enjoy a small-model property, which means that checking the validity of theory-free, quantifier-free SL entailments and WSL entailments is even decidable:

Claim 4.11. *The validity problem of theory-free, quantifier-free entailments (without SIDs) is decidable for both SL ([14]) and WSL.*

The correspondence between SL and WSL breaks down once theories, systems of inductive definitions and even limited quantification are involved, since SL becomes incomplete with the inclusion of any of them, but as we will prove in the next subsection, WSL remains complete even when all three are included.

4.3 Completeness of Effectively Determined-Heap WSL via First-Order Encoding

In this subsection we define a fragment of WSL, called Effectively Determined-Heap (EDH), where the second-order quantification over heaplets collapses to a single determined heaplet for every

formula in the context of entailments. We then give a first-order encoding of WSL entailments of EDH formulas modulo systems of inductive definitions, and show that it is *model preserving*: that is, we can capture precisely the semantics of WSL in FOL. Our FO encoding resembles existing encodings of SL (e.g., [10]). However, prior work does not show completeness properties for the encoding, and caters to restricted fragments, where SIDs are required to have distinct cases in the definition of each inductive predicate.

Definition 4.12. The fragment of Effectively Determined-Heap (EDH) formulas consists of formulas of the form $\exists \boldsymbol{y}. \bigvee_i \forall \boldsymbol{u}_i.\varphi_i$, where each φ_i is a quantifier-free *conjunctive* formula. We denote by EDH(WSL) the sub-logic of WSL, where formulas are restricted to the EDH fragment.

As mentioned above, WSL does not directly tackle the incompleteness stemming from the second-order quantification over heaplets. Instead, this source of incompleteness is eliminated by the syntactic restrictions of the EDH(WSL) fragment. EDH(WSL) is defined in a manner similar to the Effectively PRopositional (EPR) fragment of FOL, but whereas EPR only restricts the quantifier structure, EDH(WSL) also restricts disjunctions in formulas. While EPR ensures that for any formula, it suffices to consider finite structures for establishing (un)satisfiability, EDH(WSL) does not ensure that. Instead, it ensures that for any formula and structure, it suffices to consider finitely many heaplets for establishing validity of entailments. These properties make EPR a decidable fragment of FOL, while EDH(WSL) is an undecidable fragment of WSL (Thm. 4.23) that admits a complete proof system (Thm. 4.22).

Note that the restrictions imposed by the EDH fragment do not suffice to ensure completeness in SL. In particular, SL is incomplete already for theory-free, quantifier-free entailments modulo SIDs, which are included in EDH.

Remark. The EDH fragment resembles the notion of *precise predicates* (introduced in [58]), but differs in two important ways. Firstly, EDH is a syntactic fragment of formulas, whereas precision is a semantic property of predicates and formulas: using our terminology, a formula φ is precise if for any structure M and assignment v there exist at most one heaplet η such that $M, v, \eta \models \varphi$. Secondly, the two notions are incomparable. Indeed, not all precise formulas are in EDH, and the formulas we allow in EDH need not be precise, since EDH ensures that there are only finitely many satisfying heaplets but not necessarily just one. For example, the formula $\forall x.\exists y.x = y$ does not meet the syntactic requirements of EDH but it is precise. Conversely, the formula $emp \vee x \mapsto \langle nil \rangle$ is included in EDH, but it can be satisfied by two heaplets ($\emptyset$ and $\{x\}$), and therefore it is not precise.

Encoding of EDH(WSL) entailments in FOL. We first observe that we can reduce an entailment $\Gamma \vdash^{\Phi}_{\text{WSL}} \psi$ where $\Gamma \cup \{\psi\}$ consists of EDH formulas, to a finite set of entailments of the form $\varphi \vdash^{\Phi}_{\text{WSL}} \psi$ where φ is a universal conjunctive formula and ψ remains unchanged, such that $\Gamma \vdash^{\Phi}_{\text{WSL}} \psi$ is valid iff all the resulting entailments are valid. To that end, we first skolemize Γ, eliminating the existential quantifiers, and by using distributivity over the conjunction $\bigwedge \text{sk}(\Gamma)$, we translate it to an equivalent disjunction of universal conjunctive formulas. Then, by relying on the fact that $\bigvee_i \varphi_i \vdash^{\Phi}_{\text{WSL}} \psi$ is valid iff $\varphi_i \vdash^{\Phi}_{\text{WSL}} \psi$ is valid for every i, we obtain a finite set of entailments of the aforementioned form.

In the sequel we show how to encode an entailment of the form $\varphi \vdash^{\Phi}_{\text{WSL}} \exists^* \bigvee \psi_i$, where φ, ψ_i are universal, conjunctive formulas to FOL. A key idea of the translation is to (i) replace the points-to relation with unary functions $m_1, \ldots, m_n$, each of which captures one field in the record a heap location points to; and (ii) replace each inductive predicate by a pair of FO predicates, that together represent the same mapping. Recall, that in determined-heap structures, every inductive predicate P is interpreted as a partial function from tuples of elements, that satisfy the predicate, to the (unique) supporting heaplet. In the FO encoding we use P_{FO} to represent the domain of P, i.e., the

set of tuples for which the predicate holds, and P_η represents the range of P, i.e., for each tuple, the heaplet supporting its inclusion in the predicate. Instead of mapping each tuple to its supporting heaplet, P_η represents this mapping as a relation between tuples and locations in their heaplet.

Definition 4.13 (Structure correspondence). Given a vocabulary $\Sigma = \Sigma^{\text{SL}} \uplus \Sigma^{\mathcal{T}}$ and record shape $\langle \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n \rangle$ over sorts $S \uplus \{\boldsymbol{s}^*\}$, we define a FO-corresponding vocabulary $\mathcal{F}(\Sigma) = \Sigma^{\text{SL}}_{\text{FO}} \uplus \Sigma^{\mathcal{T}}$ over $S \uplus \{\boldsymbol{s}^*\}$, where Σ^{SL} and $\Sigma^{\text{SL}}_{\text{FO}}$ contain the same constant symbols; for every $i \in [n]$, $\Sigma^{\text{SL}}_{\text{FO}}$ contain a function symbol $m_i \colon \boldsymbol{s}^* \to \boldsymbol{s}_i$; and for every inductive predicate $P \in \Sigma^{\text{SL}}$ with signature $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$ there exists a pair of relations $P_{\text{FO}}, P_\eta \in \Sigma^{\text{SL}}_{\text{FO}}$ with signatures $P_{\text{FO}} \colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$, $P_\eta \colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k \times \boldsymbol{s}^*$.

We define a correspondence between WSL structures over Σ and FO structures over $\mathcal{F}(\Sigma)$. Two structures M, M_{FO} over $\Sigma, \mathcal{F}(\Sigma)$ (resp.) are said to correspond when

(1) they have the same domain $\mathcal{D}^M = \mathcal{D}^{M_{\text{FO}}}$;
(2) for any symbol $s \in \Sigma^{\mathcal{T}}$, $s^M = s^{M_{\text{FO}}}$;
(3) for any constant symbol $c \in \Sigma^{\text{SL}}$, $c^M = c^{M_{\text{FO}}}$;
(4) for any element $d \in \mathcal{D}(\boldsymbol{s}^*)$, $h^M(d) = \left\langle m_1^{M_{\text{FO}}}(d), \ldots, m_n^{M_{\text{FO}}}(d) \right\rangle$; and
(5) for any inductive predicate $P \colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$ and elements $d_1 \in \mathcal{D}^M(\boldsymbol{s}_1), \ldots, d_k \in \mathcal{D}^M(\boldsymbol{s}_k)$,

$$P^M(d_1, \ldots, d_k) = \begin{cases} \left\{ d' \;\middle|\; \langle d_1, \ldots, d_k, d' \rangle \in P_\eta^{M_{\text{FO}}} \right\} & \text{if } \langle d_1, \ldots, d_k \rangle \in P_{\text{FO}}^{M_{\text{FO}}}, \\ \bot & \text{otherwise.} \end{cases}$$

Note that every WSL structure M over Σ has a corresponding FO structure M_{FO} over $\mathcal{F}(\Sigma)$ and vice versa. Observe that corresponding structures equally satisfy the theory axiomatization, since they agree on all theory symbols, and the satisfaction relation is defined identically.

Claim 4.14. *If M, M_{FO} are corresponding structures, then $M \models \Gamma^{\mathcal{T}} \iff M_{\text{FO}} \models_{\text{FO}} \Gamma^{\mathcal{T}}$.*

Having defined the FO vocabulary and the correspondence between WSL and FO structures, we now continue with the translation of entailments modulo SID which consists of two parts: the translation of universal, conjunctive formulas, which is the basic building block of the translation, and the translation of systems of inductive definitions, which builds upon the translation of universal, conjunctive formulas.

Translation of universal, conjunctive SL formulas. Since we only consider determined-heap structures, every atomic formula has a unique heaplet (if any) that satisfies it. This property is maintained through conjunction and universal quantification, allowing us to map every universal conjunctive formula to its supporting heaplet. Accordingly, in analogy to how an inductive predicate P is captured by a pair P_{FO}, P_η, we translate every universal conjunctive SL formula φ into a pair of formulas $\varphi_{\text{FO}}, \varphi_\eta$, where φ_η records the expected heaplet of φ, and φ_{FO} captures the tuples of elements that satisfy φ, using the heaplet-recording formulas to express heap restrictions. Intuitively, each occurrence of an inductive predicate P in φ is replaced by P_{FO} in φ_{FO}, each separating conjunction adds a restriction that the expected heaplets of the two sub-formulas are disjoint, and each conjunction and universal quantifier add a restriction that the expected heaplets of the sub-formulas are equal. As for φ_η, the encoding uses the free variable x^* to capture the locations in the expected heaplet. When φ is an application of an inductive predicate P, this is done using P_η, and the other cases follow the semantics in the obvious way.

Definition 4.15. Given a universal conjunctive formula SL φ over a vocabulary Σ, we define a transformation of φ into FOL as a pair of formulas $\varphi_{\text{FO}}, \varphi_\eta$ over $\mathcal{F}(\Sigma)$, defined recursively in Table 1.

To formalize the role of φ_η as encoding the expected heaplet, we use the following definition, which defines the set of locations represented by values of x^* in satisfying assignments of φ_η.

Table 1. Translation of universal conjunctive SL formula into FOL.

φ	φ_{FO}	φ_η
$t_1 \bowtie t_2$	$t_1 \bowtie t_2$	false
$Q(t_1, \ldots, t_k)$	$Q(t_1, \ldots, t_k)$	false
$\neg Q(t_1, \ldots, t_k)$	$\neg Q(t_1, \ldots, t_k)$	false
emp	true	false
$P(t_1, \ldots, t_k)$	$P_{\text{FO}}(t_1, \ldots, t_k)$	$P_\eta(t_1, \ldots, t_k, x^*)$
$t \mapsto \langle t_1, \ldots, t_n \rangle$	$t \neq nil \wedge \bigwedge_{i=1}^{n} t_i = m_i(t)$	$x^* = t$
$\alpha \wedge \beta$	$\alpha_{\text{FO}} \wedge \beta_{\text{FO}} \wedge \forall x^*.\alpha_\eta \leftrightarrow \beta_\eta$	α_η
$\alpha * \beta$	$\alpha_{\text{FO}} \wedge \beta_{\text{FO}} \wedge \neg\exists x^*.\alpha_\eta \wedge \beta_\eta$	$\alpha_\eta \vee \beta_\eta$
$\forall x.\alpha$	$(\forall x.\alpha_{\text{FO}}) \wedge \forall x_1, x_2.\forall x^*.\alpha_\eta[x_1/x] \leftrightarrow \alpha_\eta[x_2/x]$	α_η

Definition 4.16. Given a FO structure M_{FO}, assignment v, and SL formula φ, we denote

$$[\![\varphi]\!]_v^{M_{\text{FO}}} \triangleq \left\{ d \in \mathcal{D}(\boldsymbol{s}^*) \;\middle|\; M_{\text{FO}}, v[d/x^*] \models \varphi_\eta \right\}.$$

We formalize the aforementioned properties of the translation with the following two claims.

Claim 4.17. *For any universal conjunctive formula φ, corresponding models M, M_{FO}, assignment v, and heaplet $\eta \subseteq \mathcal{D}(\boldsymbol{s}^*)$, we have that $M, v, \eta \models \varphi \Rightarrow \eta = [\![\varphi]\!]_v^{M_{\text{FO}}}$.*

Claim 4.18. *For any universal conjunctive formula, corresponding models M, M_{FO}, and assignment v, we have that $M, v, [\![\varphi]\!]_v^{M_{\text{FO}}} \models \varphi \iff M_{\text{FO}}, v \models \varphi_{\text{FO}}$.*

Translation of systems of inductive definitions. We now turn to encoding the FP semantics of systems of inductive definitions into FOL. As before, we divide the encoding into two parts: one that encodes the property that the interpretation of the inductive predicates forms a fixpoint, with respect to a SID, and the other ensures that the supporting heaplets of inductive predicates match the expected heaplets of their definitions.

Definition 4.19. For a SID Φ and inductive predicate P, let $\Phi(P) = \exists \boldsymbol{y}. \bigvee_j \rho_j^P(\boldsymbol{x}, \boldsymbol{y})$, we define the following axioms

$$\begin{aligned} \Phi_{\text{FO}}(P)(\boldsymbol{x}) &\triangleq P_{\text{FO}}(\boldsymbol{x}) \leftrightarrow \exists \boldsymbol{y}. \bigvee_j \left(\rho_j^P\right)_{\text{FO}}, \\ \Phi_\eta(P)(\boldsymbol{x}) &\triangleq \forall \boldsymbol{y}. \bigwedge_j \left(\left(\rho_j^P\right)_{\text{FO}} \rightarrow \forall x^*.P_\eta(\boldsymbol{x}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta \right). \end{aligned}$$

and define the translation of the system as

$$\Phi_{\text{FO}} \triangleq \bigwedge_{P \in \Sigma^{\text{SL}}} \forall \boldsymbol{x}.\Phi_{\text{FO}}(P) \wedge \Phi_\eta(P).$$

We formalize the correctness of the SID encoding in the following claim.

Claim 4.20. *Given a system Φ, and corresponding models M, M_{FO},*

$$M \in \mathcal{M}_{\text{FP}}(\Phi) \iff M_{\text{FO}} \models_{\text{FO}} \Phi_{\text{FO}}.$$

We now arrive at the complete encoding of entailments in EDH(WSL) modulo SIDs. Given an entailment $\varphi \vdash_{\text{WSL}}^{\Phi} \exists \boldsymbol{u}. \bigvee_i \psi_i$ the basis of the encoding is to replace each of Φ, φ and ψ_i with their FO counterparts, $\Phi_{\text{FO}}, \varphi_{\text{FO}}$ and $(\psi_i)_{\text{FO}}$. This is indeed necessary for guaranteeing the correctness of the encoding, but is not sufficient. To see this, consider the simpler case of $\varphi \vdash_{\text{WSL}} \psi$, where φ and ψ are both universal conjunctive SL formulas. As exemplified by Claim 4.18, satisfaction of ψ_{FO} by some structure M_{FO} and assignment v corresponds to satisfaction of ψ by M, v and *the expected heaplet* of

ψ, namely $[\![\psi]\!]_v^{M_{\text{FO}}}$. However, for the validity of the entailment we must have that for *any* heaplet η, if $M, v, \eta \models \varphi$ then $M, v, \eta \models \psi$. Since φ and ψ are universal conjunctive SL formulas, there is at one expected heaplet for each, and this requirement translates to the condition that for any structure and assignment that satisfy φ with its expected heaplet, they also satisfy ψ with its expected heaplet, *and both expected heaplets are the same*: $\forall x^*.\varphi_\eta \leftrightarrow \psi_\eta$. The same principle generalizes to entailments with consequents of the form $\exists \boldsymbol{u}.\ \bigvee_i \psi_i$, as shown in the following theorem.

Theorem 4.21 (WSL/FO Encoding). *A WSL entailment* $\varphi \vdash^{\Phi}_{\text{WSL}} \exists \boldsymbol{u}.\ \bigvee_i \psi_i$ *is valid iff*

$$\Gamma^{\mathcal{T}} \models_{\text{FO}} (\Phi_{\text{FO}} \wedge \varphi_{\text{FO}}) \rightarrow \exists \boldsymbol{u}.\ \bigvee_i \Big((\psi_i)_{\text{FO}} \wedge \forall x^*.\varphi_\eta \leftrightarrow (\psi_i)_\eta \Big),$$

or, equivalently, iff

$$\Gamma^{\mathcal{T}} \cup \left\{ \Phi_{\text{FO}}, \varphi_{\text{FO}}, \forall \boldsymbol{u}.\ \bigwedge_i \left(\Big(\forall x^*.\varphi_\eta \leftrightarrow (\psi_i)_\eta \Big) \rightarrow \neg (\psi_i)_{\text{FO}} \right) \right\}$$

is unsatisfiable (where $\Gamma^{\mathcal{T}}$ *is the first-order axiomatization of the theory).*

We can thus conclude that for every SID Φ , finite set of EDH(WSL) formulas Γ and EDH(WSL) formula ψ, the entailment $\Gamma \vdash^{\Phi}_{\text{WSL}} \psi$ is reducible to a FOL formula α_{FO} such that the entailment is valid (modulo theory $\mathcal{T}$) iff $\Gamma^{\mathcal{T}} \models_{\text{FO}} \alpha_{\text{FO}}$. In case Φ and all formulas are theory-free we can simply replace $\Gamma^{\mathcal{T}}$ with $\emptyset$. We now reach the culmination of the FO encoding. By having an equivalence between valid entailments in EDH(WSL) and entailments in FOL, we can carry key properties of FOL over to WSL. In particular, since $\Gamma^{\mathcal{T}}$ is a recursive set of first-order sentences, FOL has a complete proof system for checking $\Gamma^{\mathcal{T}} \models_{\text{FO}} \alpha_{\text{FO}}$. From this follows the completeness of EDH(WSL):

Theorem 4.22 (EDH(WSL) Completeness). *EDH(WSL) admits a complete proof system for entailments modulo systems of inductive definitions.*

This gives a complete but perhaps unnatural proof mechanism for EDH(WSL): translation into FOL and then using FOL proof systems. In the next section we study a more natural proof mechanism for WSL and SL: the ubiquitous fold/unfold mechanism.

Note that this completeness result does not imply decidability, and in fact EDH(WSL) is not decidable. This can be shown by encoding a tiling problem as a system of inductive definitions (e.g., predicates representing whether a coordinate in the plane is covered by some tile) and constructing an entailment that is valid exactly when there is no way to tile the top-right quadrant of the plane.

Theorem 4.23 (EDH(WSL) Undecidability). *The validity problem of EDH(WSL) entailments modulo systems of inductive definitions is undecidable.*

5 Fold/Unfold Proof Mechanisms: Soundness and Completeness

Equipped with the tools of FOL, we now turn to formalizing fold/unfold proof mechanisms of separation logic, and discussing their soundness and completeness in regards to both standard Separation Logic (SL) and Weak Separation Logic (WSL). We side-step the need for quantifier instantiations in SL proof mechanisms by restricting ourselves to theory-free SIDs and theory-free, quantifier-free formulas. Similarly to entailments in EDH(WSL), we need only consider the case $\varphi \vdash^{\Phi}_{\mathcal{L}} \bigvee_i \psi_i$ where φ, ψ_i are conjunctive formulas, since we can always replace a finite set of assumptions by a single assumption, and we can then eliminate disjunctions in the antecedent by splitting the entailment.

We show that though fold/unfold is sound for both SL and WSL, and expectedly incomplete for SL, surprisingly, it is complete for WSL. In this sense, WSL is the appropriate logic to consider when working with fold/unfold proof mechanisms.

We start by deriving a formal definition of an *abstract* fold/unfold proof mechanism, motivated by real-world tools for SL, but glossing over irrelevant details and minutia.

5.1 Abstract Fold/Unfold Proof Mechanism

The heart of a fold/unfold proof mechanism lies in its two namesake transformations: folding, which replaces a formula that matches one of the cases in a definition of an inductive predicate with its application, and unfolding, which expands an inductive predicate into its definition. Though in practice these transformations happen "in-place", in the abstract they can be thought of as two kinds of axioms added to the antecedent of the entailment. These axioms are thus the design principle behind fold/unfold proof mechanisms, whereas any specific choice of when to fold or unfold is an implementation detail. In the sequel we therefore define an abstract fold/unfold proof mechanism as a recursively enumerable set of finite sets of such axioms, where for each set of axioms, we check the validity of entailment, *modulo these axioms*. A proof is found if one of these entailments turns out to be valid.

We start with a formal definition of fold/unfold axioms.

Definition 5.1 (Fold/unfold axioms). Let Φ be a SID, $P\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$ an inductive predicate, and $\Phi(P) = \exists y_1, \ldots, y_\ell.\ \bigvee_j \rho_j^P$. Given terms $\boldsymbol{t} = \langle t_1, \ldots, t_k \rangle$ and $\boldsymbol{t'} = \langle t'_1, \ldots, t'_\ell \rangle$, a *fold axiom* of $\Phi(P)$ with terms $\boldsymbol{t}, \boldsymbol{t'}$ is the SL formula

$$\mathrm{fold}(\Phi(P), \boldsymbol{t}, \boldsymbol{t'}) \triangleq \bigwedge_j \Big(\rho_j^P[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{t'}/\boldsymbol{y}] \to P(\boldsymbol{t}) \Big).$$

Given terms $\boldsymbol{t}$ and *fresh constants* $\boldsymbol{c} = \langle c_1, \ldots, c_\ell \rangle$ an *unfold axiom* of $\Phi(P)$ with terms $\boldsymbol{t}, \boldsymbol{c}$ is the SL formula

$$\mathrm{unfold}(\Phi(P), \boldsymbol{t}, \boldsymbol{c}) \triangleq P(\boldsymbol{t}) \to \bigvee_j \rho_j^P[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{c}/\boldsymbol{y}].$$

Note that the terms $\boldsymbol{t}, \boldsymbol{t'}$ used in fold/unfold axioms can themselves include fresh constants. Importantly, even though SIDs may include existential quantifiers, observe that all fold/unfold axioms are quantifier-free (and theory-free).

Note. We extend the satisfaction relation (of Def. 4.4) to the $\to$ connective in the obvious way.

The small-model property of theory-free, quantifier-free SL formulas is maintained in the presence of the $\to$ connective. This means that Claims 4.10 and 4.11 continue to hold when the finite set of assumptions in entailments includes fold/unfold axioms. That is, $U \cup \Gamma \vdash_{\mathrm{SL}} \psi$ is valid iff $U \cup \Gamma \vdash_{\mathrm{WSL}} \psi$ is valid, and checking validity for both kinds of entailments is decidable. Decidability justifies using such entailments as the base case of fold/unfold proof mechanisms, which iteratively produce finite sets of fold/unfold axioms and add them as assumptions. Interestingly, entailments with finitely many fold/unfold axioms are equi-valid in SL and WSL, even though they are not equi-valid modulo SIDs. This discrepancy is manifested in the (in)completeness of fold/unfold mechanisms: as we show next, this procedure is complete for WSL, but not for SL.

We now give a formal definition of an abstract fold/unfold proof mechanism.

Definition 5.2 (Abstract fold/unfold proof mechanism). An abstract fold/unfold proof mechanism is a procedure that enumerates finite sets of fold/unfold axioms $U_1, U_2, \ldots$, and for each set U_i checks the validity of the entailment $\Gamma \cup U_i \vdash_{\mathrm{SL}} \psi$. The procedure returns "valid" when any such entailment is valid, and must be exhaustive, in the sense that for every finite set U of fold/unfold axioms, there exists some i s.t. $U \subseteq U_i$.

Typically, real-world fold/unfold proof mechanisms produce the sets of axioms $U_1, U_2, \ldots$ by enumerating individual axioms and accumulating them. In this case, exhaustiveness can be achieved

by ensuring that all fold/unfold axioms ultimately occur in the enumeration. Note that an abstract fold/unfold proof mechanism uses validity checks in SL; however, this may be equivalently done by validity checks in WSL since, as explained above, the two logics coincide for the entailments obtained by the addition of fold/unfold axioms.

In order to analyze the abstract fold/unfold proof mechanism we first give a formal definition for the soundness and completeness of a proof mechanism, and then establish soundness and (in)completeness of abstract fold/unfold for quantifier-free SL and WSL.

Definition 5.3 (Soundness and completeness). A proof mechanism is sound for a fragment of a logic $\mathcal{L} \in \{\text{SL}, \text{WSL}\}$ if whenever the mechanism returns "valid" for an entailment $\Gamma \vdash^{\Phi}_{\mathcal{L}} \psi$ within the fragment, then $\Gamma \vdash^{\Phi}_{\mathcal{L}} \psi$ is indeed valid. A proof mechanism is complete if whenever $\Gamma \vdash^{\Phi}_{\mathcal{L}} \psi$ is valid it halts and returns "valid".

An immediate consequence of the definition of fold/unfold axioms is that any FP structure of Φ and any heaplet satisfy all of them. From this follows the soundness of fold/unfold for both SL, where only LFP structures are considered, and WSL, where any FP structure is considered.

Theorem 5.4 (Soundness). *Any fold/unfold proof mechanism is sound for both SL and WSL.*

As opposed to soundness, the gap between SL and WSL pokes its head when completeness is considered. Namely, SL with inductive definitions is known to have no complete proof mechanism, even when only theory-free, quantifier-free formulas are considered [38]. In contrast, we show that any abstract fold/unfold proof mechanism is complete for the theory-free, quantifier-free fragment of WSL with inductive definitions.

Theorem 5.5 (Incompleteness for SL [38]). *Any abstract fold/unfold proof mechanism is incomplete for the theory-free, quantifier-free fragment of SL with theory-free SIDs.*

Theorem 5.6 (Completeness for WSL). *All abstract fold/unfold proof mechanisms are complete for the theory-free, quantifier-free fragment of WSL with theory-free SIDs.*

In the sequel we prove Thm. 5.6 by extending the encoding of Section 4.3 to relate fold/unfold axioms and quantifier instantiations of Φ_{FO} axioms, laying the ground to using Herbrand's theorem on the FOL side, and carrying the result back over to WSL.

5.2 Fold/Unfold Axioms and Quantifier Instantiations

Before delving into the translation of fold/unfold axioms to FOL let us first recall the FO axioms we defined for each inductive predicate P (Def. 4.19):

$$\Phi_{\text{FO}}(P)(\boldsymbol{x}) \triangleq P_{\text{FO}}(\boldsymbol{x}) \leftrightarrow \exists \boldsymbol{y}. \bigvee_j \left(\rho_j^P\right)_{\text{FO}} \qquad \Phi_\eta(P)(\boldsymbol{x}) \triangleq \forall \boldsymbol{y}. \bigwedge_j \left(\left(\rho_j^P\right)_{\text{FO}} \rightarrow \forall x^*. P_\eta(\boldsymbol{x}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta\right)$$

These can be equivalently expressed by the following pair of axioms:

$$\begin{aligned} \alpha_P(\boldsymbol{x}) &\triangleq P_{\text{FO}}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}. \bigvee_j \left(\left(\rho_j^P\right)_{\text{FO}} \wedge \left(\forall x^*. P_\eta(\boldsymbol{x}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta\right)\right), \\ \beta_P(\boldsymbol{x}) &\triangleq \forall \boldsymbol{y}. \bigwedge_j \left(\left(\rho_j^P\right)_{\text{FO}} \rightarrow P_{\text{FO}}(\boldsymbol{x}) \wedge \forall x^*. P_\eta(\boldsymbol{x}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta\right) \end{aligned}$$

We now define the FO encoding of fold/unfold axioms and observe that it essentially produces quantifier instantiations for the above axioms, where α_P matches $\text{unfold}_{\text{FO}}$ and β_P matches fold_{FO}.

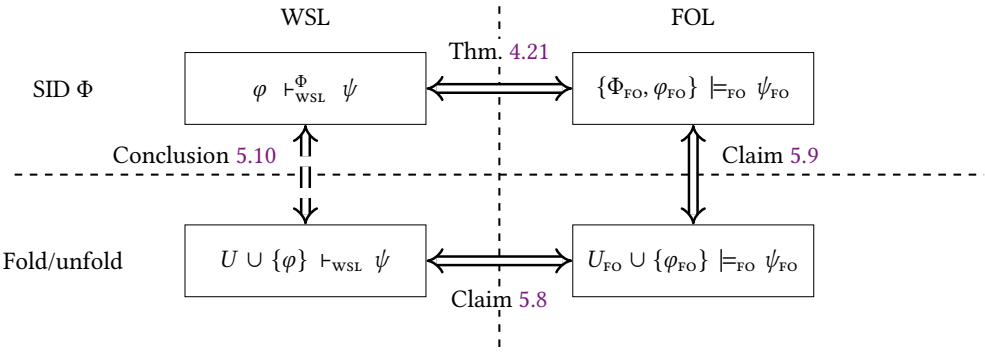


Fig. 3. Abstract fold/unfold is a complete proof mechanism for WSL: proof structure.

Definition 5.7 (Fold/unfold encoding). Given SID Φ, predicate P and terms $\boldsymbol{t}, \boldsymbol{t}'$, we define the FO encoding of the fold axiom $\text{fold}(\Phi(P), \boldsymbol{t}, \boldsymbol{t}')$ as follows

$$\text{fold}_{\text{FO}}(\Phi(P), \boldsymbol{t}, \boldsymbol{t}') \triangleq \bigwedge_j \left(\left(\rho_j^P\right)_{\text{FO}} [\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{t}'/\boldsymbol{y}] \rightarrow P_{\text{FO}}(\boldsymbol{t}) \wedge \forall x^*. P_\eta(\boldsymbol{t}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta [\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{t}'/\boldsymbol{y}] \right),$$

given terms $\boldsymbol{t}$ and fresh constants $\boldsymbol{c}$, we define the FO encoding of the unfold axiom as

$$\text{unfold}_{\text{FO}}(\Phi(P), \boldsymbol{t}, \boldsymbol{c}) \triangleq P_{\text{FO}}(\boldsymbol{x}) \rightarrow \left(\bigvee_j \left(\left(\rho_j^P\right)_{\text{FO}} [\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{c}/\boldsymbol{y}] \wedge \left(\forall x^*. P_\eta(\boldsymbol{t}, x^*) \leftrightarrow \left(\rho_j^P\right)_\eta [\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{c}/\boldsymbol{y}] \right) \right) \right),$$

and finally we lift these encodings to a set U of fold/unfold axioms, defining its encoding U_{FO} as the result of replacing each fold/unfold axiom in U with its corresponding fold_{FO} / $\text{unfold}_{\text{FO}}$.

The following claim formalizes that these definitions faithfully encode the fold/unfold axioms.

Claim 5.8. *Given corresponding structures M, M_{FO}, assignment υ, and set U of fold/unfold axioms, we have that $M_{\text{FO}}, \upsilon \models_{\text{FO}} U_{\text{FO}}$ iff $M, \upsilon, \eta \models U$ for every heaplet η.*

5.3 Bringing It Full Circle

We finally have in place all the pieces needed to show that an abstract fold/unfold proof mechanism is complete for checking the validity of theory-free, quantifier-free WSL entailments modulo theory-free SIDs. The proof leverages the FO encoding of entailments in WSL with SIDs or with fold/unfold axioms to reduce the completeness claim to a claim about entailments in FOL, where we can apply FOL results such as Herbrand's theorem and compactness. The structure of the proof is shown in Fig. 3. Considering theory-free, quantifier-free formulas and theory-free SIDs, the proof works by using a chain of three equivalences to conclude the fourth: (i) an entailment modulo SID is valid in WSL iff its encoding is valid in FOL (follows from correctness of the encoding); (ii) the encoding of an entailment modulo SID is valid in FOL iff there exists some finite set of quantifier instantiations of the system, modulo which the encoding of the entailment is valid (follows from Herbrand's theorem and compactness in FOL); (iii) the encoding of the entailment modulo a finite set of quantifier instantiations of the system is valid (in FOL) iff the entailment modulo a corresponding finite set of fold/unfold axioms is valid in WSL (follows from the correctness of the encoding); and, transitively, (iv) an entailment modulo SID is valid in WSL iff there exists a finite set of fold/unfold axioms modulo which the entailment is valid. Thus, an exhaustive set of fold/unfold axioms forms a complete proof mechanism for WSL. This line of reasoning is formalized in the following claims.

Claim 5.9. *For a theory-free SID Φ, and a pair of theory-free, quantifier-free formulas φ, ψ, we have that $\{\Phi_{\text{FO}}, \varphi_{\text{FO}}\} \models_{\text{FO}} \psi_{\text{FO}}$ iff there exists a finite set U of fold/unfold axioms for Φ s.t. $U_{\text{FO}} \cup \{\varphi_{\text{FO}}\} \models_{\text{FO}} \psi_{\text{FO}}$.*

Proof sketch. Observe that for theory-free formulas and SIDs, all applications of functions that appear in $\Phi_{\text{FO}}, \varphi_{\text{FO}}$ and ψ_{FO} are of the form $m_i(t) = t'$, where t' has no function applications. Thus, when considering the set of all ground instances of $\Phi_{\text{FO}}, \varphi_{\text{FO}}, \psi_{\text{FO}}$, per Herbrand's theorem, we need only consider ground terms with no function applications. An implication of Herbrand's theorem is that partially-ground instantiations, where an inner part of the formula stays universally quantified, is equi-satisfiable to the complete set of instantiations. Specifically, not instantiating the "same heap" condition in the encoding of the SID and fold/unfold axioms does not detract from the result of Herbrand's theorem. Therefore, the set of partial instantiations to consider correspond to the encoding of all fold/unfold axioms, and from compactness, we have a finite set of such instantiations, or equivalently fold/unfold axioms, to consider. □

Conclusion 5.10. *For a theory-free SID Φ and theory-free, quantifier-free formulas φ, ψ, the entailment $\varphi \vdash^{\Phi}_{\text{WSL}} \psi$ is valid iff there is a finite set of fold/unfold axioms U s.t. $U \cup \{\varphi\} \vdash_{\text{WSL}} \psi$ is valid.*

Recall that plain entailment of theory-free, quantifier-free formulas coincide in SL and WSL, i.e., the entailment $U \cup \{\varphi\} \vdash_{\text{WSL}} \psi$ of Conclusion 5.10 coincides with $U \cup \{\varphi\} \vdash_{\text{SL}} \psi$, which is the check used in an abstract fold/unfold mechanism. This completes the proof of Thm. 5.6.

6 Rogue Models

Imbued with our new understanding of fold/unfold as the proof mechanism of WSL (and not SL), we can now identify proof failures of fold/unfold as stemming from the semantic gap between SL and WSL, exemplified by *rogue, nonstandard* counter-models: WSL FP structures that refute validity of entailments but are not SL LFP structures intended. Soundness of fold/unfold for WSL means that these counter-models cannot be eliminated by such proof mechanisms, no matter how many efforts are invested. The ability to describe, visualize and find these models allows us to better understand tool failures, improve diagnosability of these failures, and provide guidance to the user in how to proceed with the proof (e.g., by employing induction rules, which are beyond fold/unfold). However, precisely defining a nonstandard model could be nontrivial, as counter-models to entailment in WSL may be infinite and in some cases, *must* be infinite, as we show next.

In this section we harness the FO encoding presented in Section 4.3 for the purpose of finding rogue models. We generalize from the theory-free, quantifier-free formulas, that we discussed in the previous section, to all EDH(WSL) formulas the FO encoding supports. Namely, we observe that beyond a theoretical crutch for proving the completeness of WSL, the FO encoding also gives us the ability to use existing FOL tools to show validity of entailments, in a manner similar to previous work, and to find infinite counter-models to entailments, which is unique to this work.

Rogue models. Formally, a rogue model in this context is any structure in the class of structures of WSL that is not part of the class of structures of SL. Given a SID Φ, a rogue model of Φ is any structure $M \in (\mathcal{M}_{\text{WSL}} \cap \mathcal{M}_{\text{FP}}(\Phi)) \setminus (\mathcal{M}_{\text{SL}} \cap \mathcal{M}_{\text{LFP}}(\Phi))$. The key property of the FO encoding is that it is *model preserving*. A counter-model M_{FO} to the encoded entailment corresponds to a (unique) heap-structure M that is a FP structure of the system and satisfies the antecedent of the entailment, but not the consequent. Further, M_{FO} gives us the only candidate satisfying heaplet for any sub-formula, thus resolving the second-order quantification over heaplets, and allowing us to easily verify and understand why M is a counter-model.

6.1 Heap-Reducing Systems of Inductive Definitions

Though in theory rogue models may be finite or infinite, we see that in practice systems of inductive definitions exhibit only infinite nonstandard models. As discussed in existing literature, the nature of most real-world systems of inductive definitions is such, that any recursive use of a predicate always appears under a smaller heap. If the heap is finite, then any predicate may only be unfolded a finite number of times before reaching a *base case* (i.e., a case with no recursive applications). Intuitively, this means that any tuple of elements satisfying an inductive predicate does so by virtue of a finite chain of unfoldings that reach a base case, and thus LFP and FP semantics coincide.

We follow the spirit of [10], where instead of this intuitive definition of heap-reducing, Bobot and Filliâtre [10] opt for a stricter, syntactic definition of heap-reducing.

Definition 6.1. A system of inductive definitions Φ is *heap reducing* if for every inductive predicate P, each case ρ_j^P either contains no inductive predicates, or there exists some φ s.t. ρ_j^P is equivalent to $t \mapsto \langle \ldots \rangle * \varphi$ (and all applications of inductive predicates appear in φ).

Note that restricting ourselves to heap-reducing SIDs is a mild requirement compared to stricter requirements like progress (see, e.g., [37]), used in definitions of decidable fragments of SL. Still, this heap-reducing requirement is enough to erase the distinction between FP and LFP models in heap-finite structures (as in the standard SL semantics).

Claim 6.2. *Given a heap-reducing system Φ, any heap-finite FP structure for Φ is also a LFP structure for Φ:* $\mathcal{M}_{\text{SL}} \cap \mathcal{M}_{\text{FP}}(\Phi) = \mathcal{M}_{\text{SL}} \cap \mathcal{M}_{\text{LFP}}(\Phi)$.

An immediate conclusion of the above claim is that any rogue model of Φ, that is, a model for Φ under WSL semantics but not under SL semantics, must have an infinite set of heap locations and/or be a nonstandard model of the background theory. Thus, the ability to represent and find infinite rogue models is *necessary*, as in practice all rogue models are infinite, when we limit ourselves to the standard model of the background theory.

Conclusion 6.3. *A rogue model for a heap-reducing SID Φ has to be in* $(\mathcal{M}_{\text{WSL}} \setminus \mathcal{M}_{\text{SL}}) \cap \mathcal{M}_{\text{FP}}(\Phi)$, *and if it extends the standard model of the background theory, its location domain must be infinite.*

6.2 Symbolic Structures

Rogue counter-models for entailments may be infinite, and as discussed in the previous subsection, for the very common case of a heap-reducing system, any rogue counter-model *must* be infinite. In order to find and present such models we first need a way to represent them.

One such representation is *symbolic structures* [27, 28], which were successfully used recently to find models for formulas in FOL. Symbolic structures encode infinite structures in a finite way, describing the infinite shape of the structure symbolically, with terms and formulas in Linear Integer Arithmetic (LIA). Though used primarily for uninterpreted formulas in [27, 28], symbolic structures can be naturally extended to support formulas combining uninterpreted logic with LIA, which proves useful for us, as many entailments in WSL use integers as a background theory.

We start by recalling the definition of symbolic structures, now extended to support integers. Symbolic structures use finitely many nodes to represent the elements in the domain of an explicit structure, where each node represents a set of explicit elements by a copy of some subset of the integers, specified by a formula in LIA, called the bound formula. The set of nodes form the symbolic domain of the structure (for each sort). Our extension forces the domain of sort `int` to comprise of exactly one node, representing a full copy of the integers. Interpretations of functions and relations in the explicit structure are defined symbolically over the nodes in the symbolic domain, with LIA terms and formulas that relate specific elements within the nodes.

Definition 6.4 (Extension of [27]). A *symbolic structure* for a vocabulary $\Sigma \uplus \Sigma^{\text{LIA}}$ over sorts $\{\boldsymbol{s}^*, \texttt{int}\}$ is a triple $S = (\mathcal{D}, \mathcal{B}, \mathcal{I})$, where

- $\mathcal{D}$ is the *domain* of S, mapping $\boldsymbol{s}^*$ to a *finite, non-empty* set of nodes $\mathcal{D}(\boldsymbol{s}^*)$ and mapping $\texttt{int}$ to the singleton $\mathcal{D}(\texttt{int}) = \{n_{\texttt{int}}\}$.
- $\mathcal{B}$ maps each node n to a *bound formula*, a satisfiable LIA formula $\mathcal{B}(n) \in \text{QF}_{\text{LIA}}$ with at most one free variable i, with $\mathcal{B}(n_{\texttt{int}}) = \text{true}$.
- $\mathcal{I}$ is the interpretation of symbols in $\Sigma \uplus \Sigma^{\text{LIA}}$. Each constant symbol $c\colon \boldsymbol{s}$ is interpreted by a pair $\mathcal{I}(c) = \langle n, z\rangle$ where $n \in \mathcal{D}(\boldsymbol{s})$, $[z/i] \models_{\text{LIA}} \mathcal{B}(n)$. Each function symbol $f\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k \to \boldsymbol{s}$ is interpreted as a function $\mathcal{I}(f)$ that maps nodes $n_1 \in \mathcal{D}(\boldsymbol{s}_1), \ldots, n_k \in \mathcal{D}(\boldsymbol{s}_k)$ to a pair of node $n \in \mathcal{D}(\boldsymbol{s})$ and LIA term t such that $\text{FV}(t) \subseteq \{i_1, \ldots, i_k\}$ and $\bigwedge_{j=1}^{k} \mathcal{B}(n_j)\big[i_j/i\big] \models_{\text{LIA}} \mathcal{B}(n)[t/i]$. A relation symbol $R\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$ is interpreted as a function $\mathcal{I}(R)$ that maps nodes $n_1 \in \mathcal{D}(\boldsymbol{s}_1), \ldots, n_k \in \mathcal{D}(\boldsymbol{s}_k)$ to a LIA formula $\varphi \in \text{QF}_{\text{LIA}}$ such that $\text{FV}(\varphi) \subseteq \{i_1, \ldots, i_k\}$. Additionally, for the symbols in Σ^{LIA} we require that $\mathcal{I}(0) = \langle n_{\texttt{int}}, 0\rangle$, $\mathcal{I}(1) = \langle n_{\texttt{int}}, 1\rangle$, $\mathcal{I}(+)(n_{\texttt{int}}, n_{\texttt{int}}) = \langle n_{\texttt{int}}, i_1 + i_2\rangle$, and $\mathcal{I}(<)(n_{\texttt{int}}, n_{\texttt{int}}) = i_1 < i_2$.

A symbolic structure should be understood as a finite *representation* of an explicit, potentially infinite, first-order structure, denoted by $\mathcal{E}(S)$. Formally, the explication of a symbolic structure S is a first-order structure $M = \mathcal{E}(S) = \left(\mathcal{D}^M, \mathcal{I}^M\right)$ over $\Sigma \uplus \Sigma^{\text{LIA}}$ defined as follows. For each node n in the domain of S, we define its set of explicit elements as $\mathcal{E}(n) = \{\langle n, z\rangle \mid z \in \mathcal{B}(n)\}$, where we refer to the formula $\mathcal{B}(n)$ as the set of integers $\{z \mid [z/i] \models_{\text{LIA}} \mathcal{B}(n)\}$. For each sort $\boldsymbol{s}$, its explicit domain is the set $\mathcal{D}^M(\boldsymbol{s}) = \bigcup_{n \in \mathcal{D}(\boldsymbol{s})} \mathcal{E}(n)$. For each constant symbol c, $\mathcal{I}^M(c) = \mathcal{I}(c)$. For each function symbol f and nodes $n_1, \ldots, n_k$, let $\mathcal{I}(f)(n_1, \ldots, n_k) = \langle n, t\rangle$, then for every $z_1 \in \mathcal{B}(n_1), \ldots, z_k \in \mathcal{B}(n_k)$, $\mathcal{I}^M(f)(\langle n_1, z_1\rangle, \ldots, \langle n_k, z_k\rangle) = \left\langle n, \overline{[\boldsymbol{z}/\boldsymbol{i}]}(t)\right\rangle$. For each relation symbol $R\colon \boldsymbol{s}_1 \times \cdots \times \boldsymbol{s}_k$, then $\mathcal{I}^M(R) = \left\{(\langle n_1, z_1\rangle, \ldots, \langle n_k, z_k\rangle) \mid n_j \in \mathcal{D}(\boldsymbol{s}_j),\ z_j \in \mathcal{B}(n_j) \text{ and } [\boldsymbol{z}/\boldsymbol{i}] \models_{\text{LIA}} \mathcal{I}(R)(n_1, \ldots, n_k)\right\}$.

Unlike the location sort $\boldsymbol{s}^*$, whose domain can consist of an arbitrary number of nodes and arbitrary bound formulas, we restrict the domain of the integer sort to have a single node, representing the complete set of integers, and restrict the interpretations to the standard ones. This ensures that symbolic structures extend the standard model of LIA, and thus lets us focus on rogue counter-models that arise from the other relaxations of WSL; these rogue models must be infinite in the case of heap-reducing SIDs (Conclusion 6.3).

Claim 6.5. *Given a symbolic structure S over $\Sigma \uplus \Sigma^{\text{LIA}}$, its explication $\mathcal{E}(S)$ extends the standard model of LIA, M^{LIA}.*

As shown in [27], symbolic structures admit decidable model-checking, by translating any formula φ over $\Sigma \uplus \Sigma^{\text{LIA}}$ into a pure LIA formula φ_{LIA}, such that φ is satisfied by a structure S iff φ_{LIA} is valid (recall that the validity problem of formulas in LIA is decidable).

[27] also introduced an efficient way to search for a satisfying symbolic structure for a given formula. Through the concept of a *template*, a (possibly infinite) *symbolic search space* of symbolic structures. A template specifies the general shape of a symbolic structure, by fixing its domain, and fixing finite sets of terms and formulas to be used in functions, relations and bound formulas. However, it allows these terms and formulas to include free (integer) variables, thus representing an infinite set of structures. Given such a template, a formula φ is translated into a LIA formula φ_{LIA}, such that φ_{LIA} is satisfiable iff any structure within the template satisfies φ. Moreover, the satisfying assignment to φ_{LIA} gives a complete definition of the satisfying symbolic structure.

By definition, any entailment that is valid in SL but not in WSL has a rogue, nonstandard counter-model. However, we are not guaranteed to always find a counter-model using the symbolic structures formalism. In particular, Def. 6.4 only allows to represent models that extend the standard model of the integers. This could possibly be relaxed, allowing also for encoding non-standard

models of the theory. Regardless, we can never hope to find all rogue counter-models in this way as the class of symbolic structures is recursively enumerable and model-checking is decidable, and this would therefore give us an RE invalidity procedure for EDH(WSL), which is not possible since EDH(WSL) has a validity procedure, but is undecidable (Thm. 4.23).

7 Implementation and Evaluation

We implemented the FO encoding described in Section 4.3 in a prototype tool that translates SL entailments modulo SID into first-order formulas, where a satisfying model for the FO formula represents a counter-model to entailment according to the weak semantics. We discharge these FO formulas both to Z3 [20], for checking unsatisfiability (thus proving validity) and to FEST [27] for finding rogue, potentially infinite, nonstandard counter-models. Our tool is implemented in Python [80] and receives as input Songbird [76] files that define a system of inductive definitions and an entailment check, given as two SL formulas. Our tool attempts to both prove validity (under WSL semantics) and find a counter-model, until either attempt is successful or both time out, and report the result to the user. In case of successfully finding a counter-model, our tool presents the model to the user, using FEST's syntax for describing symbolic structures.

Benchmark suite. To evaluate the effectiveness of the FO translation in eliciting rogue, nonstandard models of WSL in real-world problems, we use the SLS (Songbird with Lemma Synthesis) benchmark suite [78], which includes over 700 entailment modulo SID examples, compiled from existing benchmarks [72, 77] and extended with new examples. The SLS benchmark suite contains a variety of inductive definitions for singly- and doubly-linked lists, nested lists, list segments, trees and tree segments, with nontrivial entailment problems for each kind of system. About 30% of the examples use LIA as the background theory, while the rest use no background theories. The SLS benchmark is made up of 5 parts: (i) **SLL-VALID**, containing examples of singly-linked lists, most of which are valid with no need for induction, i.e., just by using fold/unfold, and thus also valid in WSL; (ii) **SLRD-VALID**, containing examples for singly-linked, doubly-linked lists and trees, where most are valid without induction; (iii) **SLRD-IND**, containing examples for singly-linked lists that require induction, thus invalid under WSL semantics; (iv) **SLRD-LM** *w/o integers*, containing examples for singly-linked lists, doubly-linked lists and trees, that require inductive lemmas to prove; and (v) **SLRD-LM** *w/ integers*, containing examples for singly-linked lists, doubly-linked lists and trees that require inductive lemmas, where integers are used in formulas and inductive definitions. Each part of the benchmark is further subdivided into groups, where each group defines a SID, shared between all examples in that group. Note that the benchmark does not utilize our tool to its full extent, since it is restricted to the more limited symbolic-heap fragment that most existing tools are geared at.

Implementation. Our tool uses a key insight in order to significantly simplify the FOL queries sent to Z3 and FEST, and thus improve success rates: the overwhelming majority of quantifiers used in the benchmark suite are of the form $\exists \boldsymbol{u}. \cdots * t \mapsto \langle \boldsymbol{u} \rangle * \cdots$ (for some term t), which will be translated on the FO side to $\exists \boldsymbol{u}. \cdots \wedge (\bigwedge_i m_i(t) = u_i) \wedge \cdots$. We observe that we can thus replace every occurrence of u_i with $m_i(t)$ and remove the quantifier entirely. As each quantifier is transformed into a disjunction over all nodes of the symbolic structure, this optimization allows us to tame the exponential explosion of formulas produced by FEST.

We build upon FEST's existing library of templates used for model-finding, and fine-tune a template that better captures the general shape of counter-models in separation logic. The division of the benchmark suite into groups, where each group shares a SID, allows us to inspect very few examples manually to produce this fine-tuned template.

Table 2. Evaluation results. We report for each part of the benchmark the total number of examples, the number of examples that were proved valid through the FO encoding, the number of examples where a counter-model was found, and the number of timeouts (tool failures) after 30 seconds.

Benchmark		# Examples	# Valid	# Counter-model	# Timeout
SLL-VALID		171	114	4	53
SLRD-VALID		151	85	8	58
SLRD-IND		96	0	89	7
SLRD-LM	*w/o integers*	82	5	38	38
	w/ integers	218	5	26	187

Evaluation and results. We run our tool on the SLS benchmark suite using a MacBook Pro with an Apple M1 Pro CPU and 32GB RAM, using Z3 version 4.13.3 and FEST version 2.1.1. The results are summarized in Table 2, where for each part of the benchmark we report how many examples it contains, how many were proved valid, for how many our tool produced a counter-model, and for how many our tool timed out (with a timeout of 30 seconds). We do not provide comparison with existing tools since, to the best of our knowledge, there are no other tools for finding infinite entailment-violating models in the context of SL. For the **SLL-VALID** and **SLRD-VALID** parts of the benchmark, which mostly comprise of valid entailments provable without induction, our tool is able to prove validity for ~66% and ~56% of the examples (respectively) and find counter-models for a handful of examples. As the vast majority of examples of these parts are valid under WSL semantics, we note that the failure of our tool is due to insufficient time for proving the validity of the FOL queries, rather than inability to find counter-models. For the remaining parts of the benchmark, our tool is able to find counter-models for ~93% of the examples in **SLRD-IND**, ~47% in the more complex part **SLRD-LM** *w/o integers*, and ~12% in **SLRD-LM** *w/ integers*, which uses LIA as the background theory, thus requiring the largest and most complicated symbolic structures in counter-models. We conjecture that by examining a few representative examples from the unsolved problems, the template used can be further tuned, and success rates improved.

Partial taxonomy of rogue models. We now give a partial taxonomy of the kinds of rogue models that crop up in practical SL entailments with inductive definitions. By examining the counter-models produced by our tool we identify repeating patterns and archetypes of rogues models. In essence, these archetypes correspond to the *heap shape* of the rogue model, given by the symbolic domain and interpretation of the points-to functions, with the specifics of each counter-model given by the symbolic interpretation of constants and predicates. We describe 6 such archetypes:

(1) Linked list that never reaches `null`: these are made up of two nodes n_0, n_1 in the symbolic domain, n_0 representing `null` and n_1 representing infinitely many locations that form a linked list. This archetype appears mostly in examples with unary inductive predicates over lists.
(2) Linked list that never reaches some location other than `null`: made up of three nodes n_0, n_1, n_2, where n_0 is `null`, n_1 is the unreachable location, and n_2 represents all other locations which form an infinite linked list. This archetype appears in examples with 'list-segment' predicates.
(3) Two disjoint, infinite linked lists that never reach `null`: made up of three nodes n_0, n_1, n_2 as in archetype 2, but with n_1 representing an infinite linked-list of locations. This archetype appears in more complex 'list-segment' examples and doubly-linked lists.
(4) Tree where one path never reaches `null`: similarly to archetype 1, these are made up of two nodes n_0, n_1, with n_0 representing `null`, and n_1 represents infinitely many locations on a branch

in the tree, where each points to `null` in one branch, but continues the infinite tree in the other. This archetype appears in examples with inductively-defined trees and nested linked-lists.
(5) Tree where one path never reaches some location other than `null`: made up of three nodes, akin to archetype 2, appears in examples with 'tree-segments' or simple overlaid data-structures.
(6) Trees where one path never reaches some location that never reaches `null`: akin to archetype 3, appears in more complex examples of tree segments and overlaid data-structures.

8 Related Work and Conclusion

There are several approaches to automating reasoning for SL, including the development of decidable fragments [4–6, 19, 22, 36, 59, 64, 65], search techniques over proof systems [13, 16–18, 63, 76], and translations to FOL to utilize SMT solvers and FO theorem provers [16, 17, 34, 47, 49, 50, 62, 65–68]. Our work is distinguished in this regard since it studies a rich logic (in particular, it is undecidable) with a general fragment which admits a model-preserving translation into FOL.

Fold/unfold mechanisms are extremely commonplace, with uses in logics beyond separation logic [1, 8, 11, 13, 15–17, 39, 41–44, 46, 47, 50, 51, 54, 61, 62, 68, 70, 74, 75, 81]. These are a *class* of mechanisms as opposed to a single authoritative algorithm. Our work on the (in-)completeness of these mechanisms for SL with inductive definitions is inspired by recent works [46, 51] which study the corresponding question for first-order logics. However, to our knowledge ours is the first work to develop such foundations for separation logic. This is nontrivial, since there are many more sources of incompleteness that interact in complex ways (see details in the extended version of the paper [24]). Related to our investigation of incompleteness of fold/unfold mechanisms is the work in [23], which studies incompleteness empirically for a range of automated program verifiers that work with separation logic annotations.

There is also a long tradition of studying the completeness of reasoning heuristics in the context of first-order resolution [30, 31, 33, 35, 73, 82].

Definitions with determined heaplets have been studied in early work on separation logics as *precise predicates* [5, 32, 55, 58]. There are also a plethora of works that define explicit unique heaplets for formulas in heap logics [10, 50, 52, 68].

Our technique for synthesizing rogue models is based on the recent work on symbolic structures [27]. However, there are other frameworks such as the work in [9] which studies rogue models for first-order logic formulas over algebraic datatypes generated by Isabelle/HOL. Symbolic structures are also reminiscent of abstract domains for representing unbounded structures [40, 45], but we are not aware of a formal connection between the two techniques.

We end this section with a note on inductive reasoning which can be used to overcome fold/unfold proof failures. There are several techniques for automating induction reasoning in separation logic, including deductive instantiations of induction axioms, lemma synthesis, and cyclic proofs [13, 18, 29, 78]. Using the symbolic representation of rogue models found by our technique to automatically deduce and generate missing lemmas for proving validity of entailments in separation logic is an exciting direction that we leave to future work.

In conclusion, we consider our work to be a first step towards a better understanding of SL tools and, particularly, their failures. We believe that WSL provides a way to contextualize and analyze tool failures, both on a theoretical level and as a practical framework, which we aim to pursue further in the future. While we have focused on fold/unfold reasoning, analyzing reasoning of proof mechanisms beyond fold/unfold is also of great interest and left for future work.

Data-Availability Statement

An artifact for reproducing the results is available at [25], and the latest version of the tool is available at [26].

Acknowledgments

We thank the anonymous reviewers for comments which improved the paper. The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by the Israeli Science Foundation (ISF) grant No. 2117/23.

References

[1] Nada Amin, K. Rustan M. Leino, and Tiark Rompf. 2014. Computing with an SMT Solver. In *Tests and Proofs - 8th International Conference, TAP@STAF 2014, York, UK, July 24-25, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8570)*, Martina Seidl and Nikolai Tillmann (Eds.). Springer, 20–35. doi:10.1007/978-3-319-09099-3_2

[2] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13243)*, Dana Fisman and Grigore Rosu (Eds.). Springer, 415–442. doi:10.1007/978-3-030-99524-9_24

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, 171–177. doi:10.1007/978-3-642-22110-1_14

[4] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2006. Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In *Proceedings of the 4th International Conference on Formal Methods for Components and Objects* (Amsterdam, The Netherlands) *(FMCO'05)*. Springer-Verlag, Berlin, Heidelberg, 115–137. doi:10.1007/11804192_6

[5] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2004. A Decidable Fragment of Separation Logic. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science* (Chennai, India) *(FSTTCS'04)*. Springer-Verlag, Berlin, Heidelberg, 97–109. doi:10.1007/978-3-540-30538-5_9

[6] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2005. Symbolic Execution with Separation Logic. In *Proceedings of the Third Asian Conference on Programming Languages and Systems* (Tsukuba, Japan) *(APLAS'05)*. Springer-Verlag, Berlin, Heidelberg, 52–68. doi:10.1007/11575467_5

[7] Bodil Biering, Lars Birkedal, and Noah Torp-Smith. 2007. BI-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst.* 29, 5 (2007), 24. doi:10.1145/1275497.1275499

[8] Régis Blanc, Viktor Kuncak, Etienne Kneuss, and Philippe Suter. 2013. An Overview of the Leon Verification System: Verification by Translation to Recursive Functions. In *Proceedings of the 4th Workshop on Scala* (Montpellier, France) *(SCALA '13)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. doi:10.1145/2489837.2489838

[9] Jasmin Christian Blanchette and Koen Claessen. 2010. Generating Counterexamples for Structural Inductions by Exploiting Nonstandard Models. In *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6397)*, Christian G. Fermüller and Andrei Voronkov (Eds.). Springer, 127–141. doi:10.1007/978-3-642-16242-8_10

[10] François Bobot and Jean-Christophe Filliâtre. 2012. Separation Predicates: A Taste of Separation Logic in First-Order Logic. In *Formal Methods and Software Engineering - 14th International Conference on Formal Engineering Methods, ICFEM 2012, Kyoto, Japan, November 12-16, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7635)*, Toshiaki Aoki and Kenji Taguchi (Eds.). Springer, 167–181. doi:10.1007/978-3-642-34281-3_14

[11] Robert S. Boyer and J Strother Moore. 1998. *A computational logic handbook, Second Edition.* Academic Press.

[12] Rémi Brochenin, Stéphane Demri, and Étienne Lozes. 2008. On the Almighty Wand. In *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5213)*, Michael Kaminski and Simone Martini (Eds.). Springer, 323–338. doi:10.1007/978-3-540-87531-4_24

[13] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. 2011. Automated Cyclic Entailment Proofs in Separation Logic. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6803)*, Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer, 131–146. doi:10.1007/978-3-642-22438-6_12

[14] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. 2001. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FST TCS 2001: Foundations of Software Technology and Theoretical Computer*

Science, 21st Conference, Bangalore, India, December 13-15, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2245), Ramesh Hariharan, Madhavan Mukund, and V. Vinay (Eds.). Springer, 108–119. doi:10.1007/3-540-45294-X_10

[15] Harsh Raju Chamarthi, Peter C. Dillinger, Panagiotis Manolios, and Daron Vroon. 2011. The ACL2 Sedan Theorem Proving System. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6605)*, Parosh Aziz Abdulla and K. Rustan M. Leino (Eds.). Springer, 291–295. doi:10.1007/978-3-642-19835-9_27

[16] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. 2007. Automated Verification of Shape, Size and Bag Properties. In *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS '07)*. IEEE Computer Society, USA, 307–320. doi:10.1109/ICECCS.2007.17

[17] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. 2012. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Science of Computer Programming* 77, 9 (2012), 1006–1036. doi:10.1016/j.scico.2010.07.004 The Programming Languages track at the 24th ACM Symposium on Applied Computing (SAC'09).

[18] Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. 2015. Automatic induction proofs of data-structures in imperative programs. *SIGPLAN Not.* 50, 6 (June 2015), 457–466. doi:10.1145/2813885.2737984

[19] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. 2011. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6901)*, Joost-Pieter Katoen and Barbara König (Eds.). Springer, 235–249. doi:10.1007/978-3-642-23217-6_16

[20] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 337–340. doi:10.1007/978-3-540-78800-3_24

[21] Stéphane Demri and Morgan Deters. 2015. Separation logics and modalities: a survey. *J. Appl. Non Class. Logics* 25, 1 (2015), 50–99. doi:10.1080/11663081.2015.1018801

[22] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. 2021. 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 (Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 183–199. doi:10.1007/978-3-030-79876-5_11

[23] Marco Eilers, Malte Schwerhoff, and Peter Müller. 2024. Verification Algorithms for Automated Separation Logic Verifiers. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 14681)*, Arie Gurfinkel and Vijay Ganesh (Eds.). Springer, 362–386. doi:10.1007/978-3-031-65627-9_18

[24] Neta Elad, Adithya Murali, and Sharon Shoham. 2025. Separating the Wheat from the Chaff: Understanding (In-)Completeness of Proof Mechanisms for Separation Logic with Inductive Definitions. (2025). arXiv:2511.20193 doi:10.48550/arXiv.2511.20193

[25] Neta Elad, Adithya Murali, and Sharon Shoham. 2025. Separating the Wheat from the Chaff: Understanding (In-)Completeness of Proof Mechanisms for Separation Logic with Inductive Definitions (Artifact). doi:10.5281/zenodo.17472710

[26] Neta Elad, Adithya Murali, and Sharon Shoham. 2025. Separating the Wheat from the Chaff: Understanding (In-)Completeness of Proof Mechanisms for Separation Logic with Inductive Definitions (Artifact). doi:10.5281/zenodo.17273238

[27] Neta Elad, Oded Padon, and Sharon Shoham. 2024. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification. *Proc. ACM Program. Lang.* 8, POPL (2024), 970–1000. doi:10.1145/3632875

[28] Neta Elad and Sharon Shoham. 2025. Axe 'Em: Eliminating Spurious States with Induction Axioms. *Proc. ACM Program. Lang.* 9, POPL (2025), 479–508. doi:10.1145/3704853

[29] Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. 2015. On Automated Lemma Generation for Separation Logic with Inductive Definitions. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9364)*, Bernd Finkbeiner, Geguang Pu, and Lijun Zhang (Eds.). Springer, 80–96. doi:10.1007/978-3-319-24953-7_7

[30] Harald Ganzinger and Konstantin Korovin. 2003. New Directions in Instantiation-Based Theorem Proving. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*. IEEE Computer Society, 55–64. doi:10.1109/LICS.2003.1210045

[31] Yeting Ge and Leonardo Mendonça de Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26*

- *July 2, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5643)*, Ahmed Bouajjani and Oded Maler (Eds.). Springer, 306–320. doi:10.1007/978-3-642-02658-4_25

[32] Alexey Gotsman, Josh Berdine, and Byron Cook. 2011. Precision and the Conjunction Rule in Concurrent Separation Logic. *Electron. Notes Theor. Comput. Sci.* 276 (Sept. 2011), 171–190. doi:10.1016/j.entcs.2011.09.021

[33] Fajar Haifani, Sophie Tourret, and Christoph Weidenbach. 2021. Generalized Completeness for SOS Resolution and its Application to a New Notion of Relevance. In *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12699)*, André Platzer and Geoff Sutcliffe (Eds.). Springer, 327–343. doi:10.1007/978-3-030-79876-5_19

[34] Stefan Heule, Ioannis T. Kassios, Peter Müller, and Alexander J. Summers. 2013. Verification Condition Generation for Permission Logics with Abstract Predicates and Abstraction Functions. In *ECOOP 2013 - Object-Oriented Programming - 27th European Conference, Montpellier, France, July 1-5, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7920)*, Giuseppe Castagna (Ed.). Springer, 451–476. doi:10.1007/978-3-642-39038-8_19

[35] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. 2008. On Local Reasoning in Verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963)*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer, 265–281. doi:10.1007/978-3-540-78800-3_19

[36] Radu Iosif, Adam Rogalewicz, and Jirí Simácek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7898)*, Maria Paola Bonacina (Ed.). Springer, 21–38. doi:10.1007/978-3-642-38574-2_2

[37] Radu Iosif, Adam Rogalewicz, and Jirí Simácek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7898)*, Maria Paola Bonacina (Ed.). Springer, 21–38. doi:10.1007/978-3-642-38574-2_2

[38] Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. 2014. Deciding Entailments in Inductive Separation Logic with Tree Automata. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8837)*, Franck Cassez and Jean-François Raskin (Eds.). Springer, 201–218. doi:10.1007/978-3-319-11936-6_15

[39] Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. 2011. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6617)*, Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.). Springer, 41–55. doi:10.1007/978-3-642-20398-5_4

[40] Jens Katelaan, Christoph Matheja, and Florian Zuleger. 2019. Effective Entailment Checking for Separation Logic with Inductive Definitions. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11428)*, Tomás Vojnar and Lijun Zhang (Eds.). Springer, 319–336. doi:10.1007/978-3-030-17465-1_18

[41] Matt Kaufmann and J. S. Moore. 1997. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. Softw. Eng.* 23, 4 (April 1997), 203–213. doi:10.1109/32.588534

[42] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. 2000. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, USA. doi:10.1007/978-1-4615-4449-4

[43] K. Rustan M. Leino. 2012. Automating Induction with an SMT Solver. In *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation* (Philadelphia, PA) *(VMCAI'12)*. Springer-Verlag, Berlin, Heidelberg, 315–331. doi:10.1007/978-3-642-27940-9_21

[44] K. Rustan M. Leino and Nadia Polikarpova. 2013. Verified Calculations. In *Verified Software: Theories, Tools, Experiments - 5th International Conference, VSTTE 2013, Menlo Park, CA, USA, May 17-19, 2013, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8164)*, Ernie Cohen and Andrey Rybalchenko (Eds.). Springer, 170–190. doi:10.1007/978-3-642-54108-7_9

[45] Tal Lev-Ami, Neil Immerman, Thomas W. Reps, Shmuel Sagiv, Siddharth Srivastava, and Greta Yorsh. 2005. Simulating Reachability Using First-Order Logic with Applications to Verification of Linked Data Structures. In *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3632)*, Robert Nieuwenhuis (Ed.). Springer, 99–115. doi:10.1007/11532231_8

[46] Christof Löding, P. Madhusudan, and Lucas Peña. 2018. Foundations for natural proofs and quantifier instantiation. *PACMPL* 2, POPL (2018), 10:1–10:30. doi:10.1145/3158098

[47] P. Madhusudan, Xiaokang Qiu, and Andrei Ştefănescu. 2012. Recursive Proofs for Inductive Tree Data-structures. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Philadelphia,

PA, USA) *(POPL '12)*. ACM, New York, NY, USA, 123–136. doi:10.1145/2103656.2103673

[48] John C. Mitchell and Gordon D. Plotkin. 1985. Abstract types have existential types. In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (New Orleans, Louisiana, USA) *(POPL '85)*. Association for Computing Machinery, New York, NY, USA, 37–51. doi:10.1145/318593.318606

[49] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. 2016. Viper: A Verification Infrastructure for Permission-Based Reasoning. In *Proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 9583* (St. Petersburg, FL, USA) *(VMCAI 2016)*. Springer-Verlag, Berlin, Heidelberg, 41–62. doi:10.1007/978-3-662-49122-5_2

[50] Adithya Murali, Hrishikesh Balakrishnan, Aaron Councilman, and P. Madhusudan. 2025. FO-Complete Program Verification for Heap Logics. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 106 (April 2025), 29 pages. doi:10.1145/3720447

[51] Adithya Murali, Lucas Peña, Ranjit Jhala, and P. Madhusudan. 2023. Complete First-Order Reasoning for Properties of Functional Programs. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 259 (oct 2023), 30 pages. doi:10.1145/3622835

[52] Adithya Murali, Lucas Peña, Christof Löding, and P. Madhusudan. 2023. A First-Order Logic with Frames. *ACM Trans. Program. Lang. Syst.* 45, 2, Article 7 (may 2023), 44 pages. doi:10.1145/3583057

[53] Greg Nelson and Derek C. Oppen. 1979. Simplification by Cooperating Decision Procedures. *ACM Trans. Program. Lang. Syst.* 1, 2 (oct 1979), 245–257. doi:10.1145/357073.357079

[54] Huu Hai Nguyen and Wei-Ngan Chin. 2008. Enhancing Program Verification with Lemmas. In *Proceedings of the 20th International Conference on Computer Aided Verification* (Princeton, NJ, USA) *(CAV '08)*. Springer-Verlag, Berlin, Heidelberg, 355–369. doi:10.1007/978-3-540-70545-1_34

[55] Peter W. O'Hearn. 2004. Resources, Concurrency and Local Reasoning. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3170)*, Philippa Gardner and Nobuko Yoshida (Eds.). Springer, 49–67. doi:10.1007/978-3-540-28644-8_4

[56] Peter W. O'Hearn. 2012. A Primer on Separation Logic (and Automatic Program Verification and Analysis). In *Software Safety and Security - Tools for Analysis and Verification*, Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann (Eds.). NATO Science for Peace and Security Series - D: Information and Communication Security, Vol. 33. IOS Press, 286–318. doi:10.3233/978-1-61499-028-4-286

[57] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs that Alter Data Structures. In *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2142)*, Laurent Fribourg (Ed.). Springer, 1–19. doi:10.1007/3-540-44802-0_1

[58] Peter W. O'Hearn, Hongseok Yang, and John C. Reynolds. 2004. Separation and information hiding. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, Neil D. Jones and Xavier Leroy (Eds.). ACM, 268–280. doi:10.1145/964001.964024

[59] Jens Pagel. 2020. *Decision procedures for separation logic: beyond symbolic heaps.* Ph. D. Dissertation. Technische Universität Wien.

[60] Matthew J. Parkinson and Gavin M. Bierman. 2005. Separation logic and abstraction. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, Jens Palsberg and Martín Abadi (Eds.). ACM, 247–258. doi:10.1145/1040305.1040326

[61] Grant O. Passmore, Simon Cruanes, Denis Ignatovich, Dave Aitken, Matt Bray, Elijah Kagan, Kostya Kanishev, Ewen Maclean, and Nicola Mometto. 2020. The Imandra Automated Reasoning System (System Description). In *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12167)*, Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.). Springer, 464–471. doi:10.1007/978-3-030-51054-1_30

[62] Edgar Pek, Xiaokang Qiu, and P. Madhusudan. 2014. Natural Proofs for Data Structure Manipulation in C Using Separation Logic. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14)*. ACM, New York, NY, USA, 440–451. doi:10.1145/2594291.2594325

[63] Juan Antonio Navarro Pérez and Andrey Rybalchenko. 2011. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, Mary W. Hall and David A. Padua (Eds.). ACM, 556–566. doi:10.1145/1993498.1993563

[64] Juan Antonio Navarro Pérez and Andrey Rybalchenko. 2013. Separation Logic Modulo Theories. In *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8301)*, Chung-chieh Shan (Ed.). Springer, 90–106. doi:10.1007/978-3-319-03542-0_7

[65] Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2013. Automating Separation Logic Using SMT. In *Proceedings of the 25th International Conference on Computer Aided Verification* (Saint Petersburg, Russia) *(CAV'13)*. Springer-Verlag,

Berlin, Heidelberg, 773–789. doi:10.1007/978-3-642-39799-8_54

[66] Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2014. Automating Separation Logic with Trees and Data. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, 711–728. doi:10.1007/978-3-319-08867-9_47

[67] Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2014. GRASShopper - Complete Heap Verification with Mixed Specifications. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8413)*, Erika Ábrahám and Klaus Havelund (Eds.). Springer, 124–139. doi:10.1007/978-3-642-54862-8_9

[68] Xiaokang Qiu, Pranav Garg, Andrei Ştefănescu, and P. Madhusudan. 2013. Natural Proofs for Structure, Data, and Separation. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. ACM, New York, NY, USA, 231–242. doi:10.1145/2491956.2462169

[69] John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*. IEEE Computer Society, 55–74. doi:10.1109/LICS.2002.1029817

[70] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. 2008. Liquid Types. *SIGPLAN Not.* 43, 6 (jun 2008), 159–169. doi:10.1145/1379022.1375602

[71] Mihaela Sighireanu. 2021. SL-COMP: Competition of Solvers for Separation Logic: Report on the Third Edition. *Int. J. Softw. Tools Technol. Transf.* 23, 6 (dec 2021), 895–903. doi:10.1007/s10009-021-00628-w

[72] Mihaela Sighireanu and David R. Cok. 2014. Report on SL-COMP 2014. *J. Satisf. Boolean Model. Comput.* 9, 1 (2014), 173–186. doi:10.3233/SAT190107

[73] Mark E. Stickel. 1985. Automated Deduction by Theory Resolution. *J. Autom. Reason.* 1, 4 (1985), 333–355. doi:10.1007/BF00244275

[74] Philippe Suter, Mirco Dotta, and Viktor Kunćak. 2010. Decision Procedures for Algebraic Data Types with Abstractions. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Madrid, Spain) *(POPL '10)*. ACM, New York, NY, USA, 199–210. doi:10.1145/1706299.1706325

[75] Philippe Suter, Ali Sinan Köksal, and Viktor Kuncak. 2011. Satisfiability Modulo Recursive Programs. In *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6887)*, Eran Yahav (Ed.). Springer, 298–315. doi:10.1007/978-3-642-23702-7_23

[76] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9995)*, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). 659–676. doi:10.1007/978-3-319-48989-6_40

[77] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9995)*, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). 659–676. doi:10.1007/978-3-319-48989-6_40

[78] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2018. Automated lemma synthesis in symbolic-heap separation logic. *Proc. ACM Program. Lang.* 2, POPL (2018), 9:1–9:29. doi:10.1145/3158097

[79] Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. (1955).

[80] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA. doi:10.5555/1593511

[81] Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan G. Scott, Ryan R. Newton, Philip Wadler, and Ranjit Jhala. 2018. Refinement reflection: complete verification with SMT. *Proc. ACM Program. Lang.* 2, POPL (2018), 53:1–53:31. doi:10.1145/3158141

[82] Lawrence Wos, George A. Robinson, and Daniel F. Carson. 1965. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *J. ACM* 12, 4 (oct 1965), 536–541. doi:10.1145/321296.321302

Received 2025-07-10; accepted 2025-11-06